

PROJET INFORMATIQUE :

ARITHMÉTIQUE

MULTIPRÉCISION

Groupe TD C

I- SOMMAIRE

I- SOMMAIRE	1
II- Introduction	3
A) Affiche_lentier	4
B) lAdjust	5
C) lAdjust_realloc	6
IV- Fonctions mult_classique, mul_mod	9
A) Mult_classique	10
B) Mul_mod	13
V- Fonction add_lentier	14
VI- Fonctions cmp_lentier, sub_lentier	18
A) Cmp_lentier	18
B) sub_lentier	20
C) Allonge_lentier	21
VII- Fonction exp_mod	22
VIII- Fonction dec2lentier	25
A) dec2lentier	27
B) Fonction d'initialisation d'un lentier	29
C) Fonction dec2lentier finale	30
IX- Fonction div_eucl	36
A) B2BRightShift	37
B) B2BLeftShift	38
C) W2WLeftShift	39
D) div_eucl_QR	40
E) div_eucl	41
X- Fonction lentier2dec	54
XI- Organisation et conclusion	60

II- Introduction

Pour la fin du deuxième semestre de première année de l'IUT, le groupe de TD C de Télécom Saint-Etienne a été amené à produire un programme informatique en C++ sur le thème de l'arithmétique multiprécision. Nous allons vous présenter ce projet à travers ce compte rendu en reprenant chacun de nos codes, nos algorithmes, les problématiques rencontrées et les solutions étudiées.

Pour chacune des fonctions, on expliquera leur rôle, les éventuels paramètres et types de retour. On précisera également les problématiques rencontrées par chaque groupe, les solutions étudiées et les choix de chaque groupe.

Chaque fonction est rédigée sous forme algorithmique, chacune avec son lexique et son algorithme local.

Enfin, nous utiliserons un programme principal fourni par Monsieur Florent Bernard pour tester le bon fonctionnement de chacune des fonctions.

III- Fonctions Affiche_lentier, lAdjust, lAdjust_realloc

A) Affiche_lentier

Rôle : Affiche un entier long passé en paramètre (type lentier)

Entrée : l'entier long de type lentier

Sortie : vide

Déclaration de fonction : Affiche_lentier : la fonction(a : l'entier) => vide

Lexique local :

i : 1 entier non signé

Algorithme local :

Début

```
i <= 0
Ecrire( “ { ” )
Tant que i < a.size
    Ecrire ( ai )
    i <= i +1
FinTantque
Ecrire ( “ } ” )
```

Fin

B) lAdjust

Rôle : la taille est ajustée, les mots machines non utilisés restent en mémoire et ne sont pas utilisés

Entrée : l'entier long type de type lentier

Sortie : vide

Déclaration de fonction : lAdjust : la fonction(a : l'entier) => vide

Lexique local :

i : 1 entier non signé

Algorithme local :

Début

```
i <= a.size
Tant que i > 0
Si    *&ai - 1 = 0
    Alors
        a.size <= a.size -1
        i <= i - 1
    Sinon
        i <= 0
FinSi
```

Fin

C) lAdjust_realloc

Rôle : La taille est réajustée et une nouvelle zone de cette taille doit être allouée dans laquelle seront copiés tous les chiffres significatifs du lentier.

Entrée : l'entier long de type de lentier

Sortie : vide

Déclaration de fonction : lAdjust_realloc : la fonction (a : l'entier) => vide

Lexique local :

i : 1 entier non signé

Algorithme local :

Début

```
i <= 0
lentier new_a
new_a.size <= a.size
new_a.p = allouer (a,unsigned int)
Tant que i < a.size
    new_a[i] <= a[i]
FinTantque
Libérer (new_a.p)
a <= new_a
```

Fin

Vecteurs de tests:

Nous avons testé sur un main la fonction Affiche_lentier ou on initialise des valeurs à un tableau de manière aléatoire

```
#include<iostream>
#include"mes_fonctions.h"
using namespace std;
int main()
{
    lentier b;

    unsigned int tab[5];
    tab[0] = 0;
    tab[1] = 12;
    tab[2] = 4;
    tab[3] = 3;
    tab[4] = 0;

    b.p = &tab[0];
    b.size = 5;

    Affiche_lentier(b);

    return 0;
}
```

```
{0, 12, 4, 3, 0, }

Sortie de C:\Users\johan\source\repos\Project2\Debug\Project2.exe (processus 23452). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

Pour tester notre fonction nous avons utilisé un tableau ou on initialise des valeurs. Nous avons alloué des valeurs du tableau à 0 afin de mieux tester les fonctions lAdjust et lAdjust_realloc.

```
#include<iostream>
#include"mes_fonctions.h"
using namespace std;
int main()
{
    lentier b;

    unsigned int tab[5];
    tab[0] = 0;
    tab[1] = 12;
    tab[2] = 6;
    tab[3] = 0;
    tab[4] = 0;

    b.p = &tab[0];
    b.size = 5;

    lAdjust(b);
    Affiche_lentier(b);

    return 0;
}
```

Ici, le nombre est bien affiché. De plus, les valeurs de , tab[3] et tab[4] ont été supprimées.
Donc notre fonction a bien été ajustée et écrit notre chiffre.

Nous avons fait de même pour la fonction lAdjust_realloc.

```
{0, 12, 6, }

Sortie de C:\Users\johan\source\repos\Project2\Debug\Project2.exe (processus 8824). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

```
#include<iostream>
#include"mes_fonctions.h"
using namespace std;
int main()
{
    lentier b;

    unsigned int tab[5];
    tab[0] = 0;
    tab[1] = 12;
    tab[2] = 4;
    tab[3] = 3;
    tab[4] = 0;

    b.p = &tab[0];
    b.size = 5;|
```

```
lAdjust_realloc(b);
Affiche_lentier(b);

return 0;
}
```

```
{0, 12, 4, 3, }
```

```
Sortie de C:\Users\johan\source\repos\Project2\Debug\Project2.exe (processus 8824). Code : 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre. . .
```

Ici, la valeur de tab [4] a été ajustée et le tableau a bien été affiché.

IV- Fonctions mult_classique, mul_mod

Notre groupe était chargé de réaliser deux fonctions du projet, qui sont les suivantes : mult_classique et mul_mod. Nous avons commencé lors des vacances d'avril et nous nous sommes mis d'accord pour traiter chacune des fonctions ensemble au lieu d'une chacun, nous avons échangé quasiment tous les jours pour cela. On a alors commencé par mult_classique. Celle-ci a pour but de multiplier deux entiers, entrés par l'utilisateur, et de retourner le résultat par la suite. Ainsi, nous avons eu besoin de la fonction Affiche_lentier traitée plus haut. Après quelques semaines et une réunion avec M.BERNARD, nous avons finalement réussi à achever le code de mult_classique. Par la suite, il nous manquait plus que le code de mul_mod. Son but est de multiplier deux entiers puis réduire le résultat au modulo N (ce qui correspond à une division euclidienne). Ainsi, si mult_classique est juste alors mul_mod l'est forcément aussi. Ceci explique pourquoi notre groupe ne l'a pas testée.

Voici les algorithmes, les codes commentés et les tests possibles grâce aux vecteurs de tests fournis:

A) Mult_classique

Rôle : multiplie deux lentier a et b de tailles respectives t et n et retourne un lentier r, résultat de la multiplication.

Entrée : 2 lentier a et b.

Sortie : 1 lentier résultat de $a * b$, de taille $n * t$.

Déclaration de fonction : mult_classique : la fonction ($a, b : 2$ lentier) => 1 lentier
Allouer(1 entier, 1 type) => 1 adresse vers type

Afin de réaliser la multiplication multiprécision, nous nous sommes basés sur l'algorithme suivant :

1. Pour i de 0 à $n + t - 1$ faire
 $w_i \leftarrow 0$
 FinPour
2. Pour i de 0 à $n - 1$ faire
 - (a) $c \leftarrow 0$
 - (b) Pour j de 0 à $t - 1$ faire
 $\text{temp} \leftarrow w_{i+j} + a_i \times b_j + c$
 $w_{i+j} \leftarrow \text{temp}$ reste r
 $c \leftarrow \text{temp}$ div r
 FinPour
 - (c) $w_{i+t} \leftarrow c$
 FinPour
3. Retourne $(w_{n+t-1} \dots w_1 w_0)_r$

Lexique local :

i, j, n, t, c : 5 entiers
 temp, r : 2 entiers sur 64 bits
 w : 1 lentier

Algorithme local :

Début

```

n <= a.size
t <= b.size
w.p <= Allouer(n+t, entier)
i <= 0
j <= 0
w.size <= n + t

```

Tant que $i = n + t - 1$

```
wi <= 0
Fin Tant que
i <= 0
Tant que i = n - 1
    c <= 0
    j <= 0
    Tant que j = t - 1
        temp <= wi + j + ai x bj + c
        wi + j <= temp
        c <= temp >> 32
        j <= j + 1
    Fin Tant que
    wi + t <= c
    i <= i + 1
Fin Tant que
Fin
```

Algorithme en C:

```

lentier mult_classique(lentier a, lentier b)
{
    unsigned int i, j, n, t, c; //déclaration des variables i, j, n, t, c sur 16 bits.
    unsigned int long long temp, r; //déclaration des variables temp et r sur 64 bits.
    lentier w;//déclaration d'un lentier qui va être le résultat de la multiplication multiprécision.
    n = a.size;// on associe à n la taille du lentier a.
    t = b.size;// on associe à t la taille du lentier b.
    w.p = new unsigned int[n + t];// on alloue un espace de taille (n+t) pour le pointeur w.
    i = 0;// on initialise i à 0.
    j = 0;// on initialise j à 0.
    w.size = n + t; // on initialise la taille du lentier w à (n+t).

    while (i <= n + t - 1) // boucle Tant que i = n+t-1 faire...
    {
        w.p[i] = 0;// on met la valeur du pointeur du lentier w en i à 0
        i = i++;// on incrémente i.
    }
    i = 0;// on réinitialise i à 0.
    while (i <= n - 1)// boucle Tant que i = n-1 faire...
    {
        c = 0;//on initialise c à 0.
        j = 0;//on réinitialise j à 0.
        while (j <= t - 1)// à l'intérieur de l'autre boucle, boucle Tant que j = t-1 faire...
        {
            temp = ((unsigned long long int)a.p[i])*b.p[j] + w.p[i + j] + c; /* on calcule la valeur de temp = (valeur du pointeur du lentier a en i)*(valeur du pointeur du lentier b en j)
            + (valeur du pointeur du lentier w en i+j) + la valeur de c. */
            w.p[i + j] = temp; // on affecte la variable temp au pointeur du lentier w en i+j.
            c = temp >> 32; // on affecte à la variable c la valeur de temp décalée vers la droite de 32 bits.
            j++;// on incrémenté j.
        }
        w.p[i + t] = c; // on affecte la valeur de la variable c au pointeur du lentier w en i+t.
        i++;// on incrémenté i.
    }
    return w;// on retourne la valeur du lentier w.
}

```

Test de la fonction mult_classique à l'aide d'un des vecteurs tests fournis :

```

projet_final_mult_info2          (Portée globale)          main()
1 #include<iostream>
2 #include"mult_classique.h"
3 using namespace std;
4 int main()
5 {
6     unsigned int a[10] = { 1317422656,2105073766,338470606,1116885534,253159727,1726155252,832312496,1743004898,1170536359,976937719 };
7     unsigned int b[5] = { 129991036,449434451,2030267224,434751496,1950190463 };
8     lentier aa;
9     aa.p = a;
10    aa.size = 10;
11    lentier bb;
12    bb.p = b;
13    bb.size = 5;
14    lentier res;
15    res = mult_classique(aa, bb);
16    Affiche_lentier(res);
17    delete[]res.p;
18    system("pause");
19    return 0;
20 }

C:\Users\lili\source\repos\projet_final_mult_info2\Debug\projet_final_mult_info2.exe
3694151424,2982807545,1927547140,2299576432,757373056,1848212088,2141363832,2162734159,1507005504,2698987118,1886139145,
2999857725,347308699,48277751,443592347
Appuyez sur une touche pour continuer...

```

B) Mul_mod

Rôle : multiplie deux lentier a et b de tailles respectives t et n et calcule le reste de la division du produit par un lentier N de taille k, et retourne un lentier r, le résultat.

Entrée : 3 lentier a, b et N.

Sortie : 1 lentier résultat de $(a * b) \% N$, de taille k.

Déclaration de fonction : mul_mod : la fonction (a, b, N : 3 lentier) => 1 lentier

Lexique local :

a, b, N, c, d <= 5 lentier

Algorithme local :

Début

 d <= mult_classique(a, b)

 c <= div_eucl(d, N)

 Retourne d

Fin

V- Fonction add_lentier

Rôle : additionne deux lentier a et b de tailles respectives a.size et b.size et retourne un nouveau lentier, résultat de l'addition a+b

Entrée : deux lentier a et b de tailles respectives a.size et b.size

Sortie : un nouveau lentier résultat de l'addition a+b

Déclaration de fonction : add_lentier : la fonction (a, b : 2 lentier) => 1 lentier

Lexique local :

c, i : 2 entiers
r : 1 entier sur 64 bits
s : 1 lentier

Algorithme local :

```

Début
    c <= 0
    i <= 0
    Si a.size < b.size
        Alors
            s.p <= Allouer (b.size, entier)
            Tant que i ≤ b.size - 1
                Si i < a.size
                    Alors
                        si <= (ai + bi) + c reste r
                        Si (ai + bi + c) > r
                            Alors
                                c <= 1
                            Sinon
                                c <= 0
                        Fin Si
                    Sinon
                        si <= bi
                    Fin Si
                i <= i + 1
            Fin Tant que
            sb.size <= c
            s.size <= b.size
        Sinon
            s.p <= Allouer (a.size, entier)
            Tant que i ≤ a.size - 1
                Si i < b.size
                    Alors
                        si <= (ai + bi + c) reste r
                        Si (ai + bi + c) > r

```

```
Alors
    c <= 1
Sinon
    c <= 0
    Fin Si
    Sinon
        s_i <= a_i
        Fin Si
        i <= i + 1
    Fin Tant que
    s_a.size <= c
    s.size <= a.size
Fin Si
Retourner s
Fin
```

Algorithme en C:

```
/*Dans ce code nous additionnerons les deux lentier jusqu'à ce qu'un des deux soit nul en faisant attention à la retenue.
Une fois l'un des deux nul, nous nous contenterons de recopier la fin du non-nul sur les cases de poids forts*/
lentier add_lentier(lentier a, lentier b)
{
    unsigned int c, i; //Déclaration de la variable de retenue et de comptage
    unsigned int long long r = pow(2, 32); //Déclaration et initialisation de la constante qui sert à savoir si il y a une retenue
    lentier s; //Déclaration du lentier final
    c = 0; // retenue à 0
    i = 0; // comptage à 0
    (s.p) = new unsigned int[32]; //allocation d'espace pour le resultat de l'add

    if (a.size < b.size) // Si b est plus long/grand que a
    {
        while (i <= b.size - 1) // Tant qu'on a pas parcouru tout le lentier
        {
            if (i < a.size) // tant que la case de A de rang i n'est pas nul, on l'ajoute à B et C
            {
                s.p[i] = (a.p[i] + b.p[i] + c) % r; // La case i du lentier final prend la valeur de l'addition des cases i de a, b et la retenue éventuelle
                c = ((a.p[i] + b.p[i] + c) > r) ? c = 1 : c = 0; // On vérifie si la retenue vaut 1 ou 0 et on actualise
            }
            else // si la case de A de rang i est nul
            {
                s.p[i] = b.p[i]; // Les dernières cases du lentier final prennent seulement la valeur des cases de b car a est nul
            }
            i++;
        }
        s.p[b.size] = c; // La case de poids fort de la solution prend la valeur de la dernière retenue
        s.size = b.size + 1; // Le lentier final aura une case de plus pour la retenue éventuelle
    }
    else // même chose avec A plus grand lentier
    {
        while (i <= a.size - 1)
        {
            if (i < b.size) // tant que le bit de B de rang i n'est pas nul
            {
                s.p[i] = (a.p[i] + b.p[i] + c) % r;
                c = ((a.p[i] + b.p[i] + c) > r) ? c = 1 : c = 0;
            }
            else // si le bit de B de rang i est nul
            {
                s.p[i] = a.p[i];
            }
            i++;
        }
        s.p[a.size] = c; // Le bit de poids fort prend la valeur de la dernière retenue
        s.size = a.size + 1;
    }
    return s; // On retournera la solution sous la forme du type composé lentier
}
```

Vecteurs de test:

```
int main()
{
    unsigned int a[10] = { 1137211702,155462529,1887576678,236410224,28011278,898287641,1608088896,945189868,614471277,87057456 };
    unsigned int b[11] = { 1475396884,33971349,548255716,903755255,643576929,59111273,1843312790,1498549280,332810752,547847081,177662597 };
    lentier aa;
    aa.p = a;
    aa.size = 14;
    lentier bb;
    bb.p = b;
    bb.size = 11;
    lentier res;
    res = add_lentier(aa, bb);
    Affiche_lentier(res);
    /* Résultat attendu: {2612608586,189433878,2435832394,1140165479,671588207,957398914,3451401686,2443739148,947282029,634904537,1834454827,1749627740,1893361744,1938231846,0};*/
    delete[]res.p;
    system("pause");
    return 0;
}
C:\info iut\projet\projet_info\Debug\projet_info.exe
{2612608586,189433878,2435832394,1140165479,671588207,957398914,3451401686,2443739148,947282029,634904537,3613636433,3263896416,19921780,1391347,0}
Appuyez sur une touche pour continuer... ■
```

VI- Fonctions cmp_lentier, sub_lentier

Notre groupe avait pour tâche de réaliser deux fonctions du projet : cmp_lentier, et sub_lentier. Après notre première réunion (en distanciel, sur le serveur discord) nous avons réalisé qu'il serait plus simple pour nous de coder ces fonctions en utilisant une autre : que nous avons nommée Ajuste_lentier. Son but devait être d'ajuster la taille de deux lentiers, pour pouvoir effectuer une soustraction ensuite. Nous avons décidé que, dans un premier temps, et pour commencer la création des fonctions, chacun de nous avancerait sur une des trois. Oscar Cizeron a donc travaillé en premier lieu sur la fonction Ajuste_lentier, Sophie Trouillot sur sub_lentier, et Jean-Baptiste Martin sur cmp_lentier. Nous avons décidé de notre affectation sur les fonctions en fonction de celui ou celle qui semblait le mieux comprendre le fonctionnement attendu des fonctions.

Nous avons donc commencé les esquisses de chaque fonction et, quelques jours après, nous avons mis en relation notre travail. Chacun a expliqué aux autres ses idées, problèmes, et les éventuelles solutions trouvées. Suite à cela, les autres essayaient d'aider le groupe, et d'améliorer les fonctions. Cela se tenait également sur discord. Nous avons réalisé cela plusieurs fois, jusqu'à ce que toutes les fonctions soient opérationnelles, et que nous ne trouvions plus rien à améliorer.

Voici les fonctionnements et les algorithmes de nos fonctions :

A) Cmp_lentier

Rôle : Comparer deux lentiers, et retourner un char en fonction de cette comparaison

Entrée : Deux lentier a et b

Sortie : Un caractère char

Déclaration de fonction : char cmp_lentier (a et b deux lentier)

Lexique local :

x : un caractère
taille_decremente, va, vb : trois entiers

Algorithme local :

Début

a <= lajust_realloc(a)
b <= ladjust_realloc(b)

Si(a.size > b.size)

Alors x <= 1

Fin si

Si (a.size < b.size)

Alors x <= (-1)

Fin si

Si (a.size = b.size)

taille_decremente = a.size

Alors tant que taille_decremente > 0

 va <-*(a.p + taille_decremente - 1)
 vb <-*(b.p + taille_decremente - 1)

Si (va >vb)

 Alors x = 1

 taille_decremente = 0

 Fin si

 Si (va >vb)

 Alors x = 1

 taille_decremente = 0

 Fin si

 Si (va = vb)

 Alors x = 0

 taille_decremente <= taille_decremente -1

 Fin si

Fin si

Retourner x

Fin

Cette fonction comporte deux lentier en entrée, et un caractère (char) en sortie. Le but de cette fonction est de comparer deux lentier : a et b, où a est la première entrée de la fonction. Si a est supérieur à b, la valeur du caractère de sortie est 1, si b est supérieur à a, ce caractère de sortie est -1, et enfin si a est égal à b, ce paramètre a pour valeur 0.

En réalisant cette fonction, nous avons réalisés que nous pouvions utiliser deux paramètres pour comparer la valeur de a et de b :

La taille respective des lentier (ici respectivement a.size et b.size)

La valeur respective de tous les mots des lentier.

Nous avons, après l'entrevue de moitié de projet avec Monsieur Florent Bernard, Nous nous sommes rendus compte que la meilleure méthode pour comparer efficacement les deux lentier, est de d'abord comparer leur taille, et ensuite si elle est identique, de comparer les valeurs respectives des mots des lentier en commençant par les mots de poids fort.

B) sub_lentier

Rôle : Soustraire deux lentier

Entrée : Deux entiers a et b tels que $a \geq b$

Sortie : un lentier, le résultat de la soustraction

Déclaration de fonction : sub_lentier : la fonction (a, b : 2 lentier) -> 1 lentier

Lexique local :

c : 1 entier
i, n : 2 entiers non signés
s : 1 lentier
pa, pb, ps : 3 pointeurs d'unsigned int

Algorithme local :

Début

c <- 0

i <- 0

pa <- a.p
pb <- b.p

|Adjust (a)
|Adjust (b)

Si (a.size > b.size)

Alors

b <- Allonge_lentier (b, a.size)

FinSi

n <- a.size

ps <- Allouer (n,entier non signé)

Tant que i < n

((ps+i)^) <- ((pa+i)^) - ((b.p+i)^) - c reste 2^{32}

Si (((pa+i)^) ≥ ((b.p+i)^))

Alors

c <- 0

Else

c <- 1

FinSi

i <- i +1

FinPour

s.size <- n

s.p <- ps ;

|Adjust (s)

Retourner s

Fin

C) Allonge_lentier

Rôle : augmenter la taille d'un lentier et remplir ses nouvelles cases mémoires de 0

Entrée : le lentier à allonger et la taille jusqu'à laquelle il faut allonger

Sortie : un lentier

Déclaration de fonction : lentier Allonge_lentier(lentier x, unsigned int size)

Lexique local :

z : un lentier
i : un unsigned int
j : un pointeur d'unsigned int

Algorithme local :

Début

i <= 0
Tant que i < size
 (j+i)^ <= 0
 i <= i+1
Fin Tant que

i <= 0
Tant que i < x.size
 (j+i)^ <= (x.p+i)^
 i <= i+1
Fin Tant que

z.size <= size
z.p <= j
Retourner z

Fin

VII- Fonction exp_mod

Rôle : Calcul $a^x \bmod N$ où a , x et N sont de très grands entiers

Entrée : a , x , N : 3 entier

Sortie : P : 1 entier

Déclaration de fonction : `exp_mod : la fonction (a, x, N) => p`

Lexique local :

P , temp : 2 entier

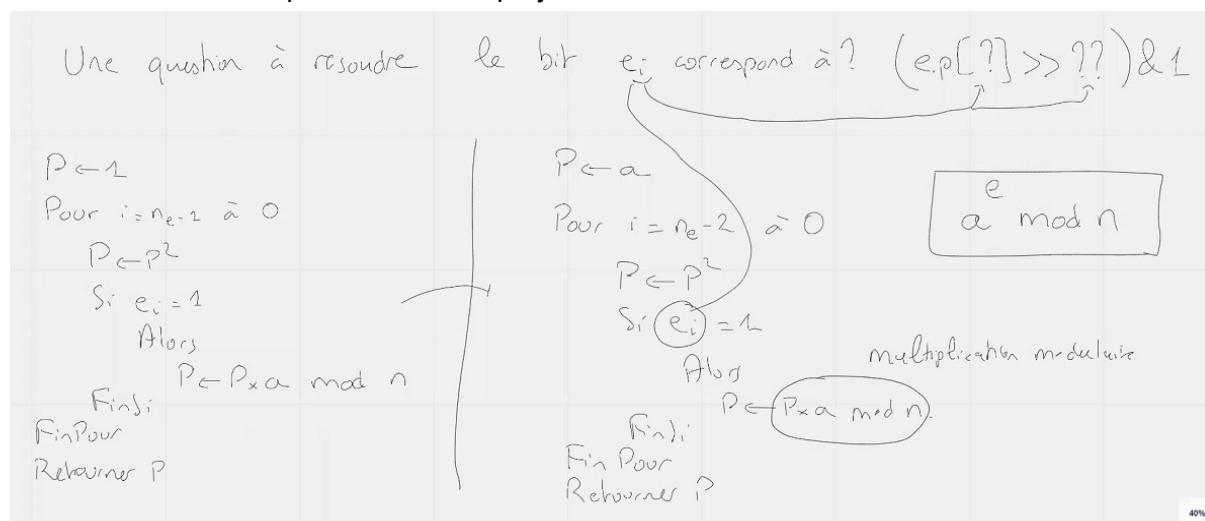
Algorithm local :

Début

```
P.p=allouer(P.size,entier)
i <= 0
Tant que i < a.size
    Pi <= ai
    i <= i + 1
Fin Tant Que
i <= 32*x.size
Tant Que i >= 0
    temp <= mul_mod(P, P, N)
    libérer(P.p)
    p <= temp
    Si x(i div 32) ET (1<<( i reste 32))
        temp = mul_mod(P, a, N)
        libérer(P.p)
        P <= temp
    Fin Si
Fin Tant Que
Retourner P
```

Fin

L'essentiel de notre travail a été la traduction de l'algorithme "Square And Multiply" présent dans le document de présentation du projet.



Pour ce faire, nous nous sommes appuyés sur les deux pistes données par M. Florent Bernard lors d'une séance de questions-réponses. Nous avions deux possibilités : initialiser P à 1 et faire varier i de $n_e - 1$ à 0 ou initialiser P à la valeur "a" et faire varier i de $n_e - 2$ à 0. En effet la condition " $x_i \text{ div } 32$ ET $(1 << (i \text{ reste } 32))$ " est toujours vraie à la première itération et P prendra finalement la valeur "a" à la fin de la première itération de la boucle "Tant Que". Dès lors, choisir la deuxième option est la plus viable d'un point de vue optimisation dans la mesure où cela nous permet d'économiser une itération tout en aboutissant à un résultat rigoureusement similaire. Ensuite nous avons fait le choix d'utiliser une variable temporaire Temp. Celle-ci nous permet d'économiser de la mémoire en travaillant avec un espace mémoire de façon temporaire avant de supprimer ce qu'il contient afin d'économiser de la RAM.

Code en C++ de la fonction:

```

lentier exp_mod(lentier a, lentier x, lentier N) {
    lentier P;
    P.size = N.size;
    P.p = new unsigned int[P.size];
    lentier temp;

    for (unsigned int i = 0; i < a.size; i++)
    {
        P.p[i] = a.p[i];
    }

    for (int i = 32 * x.size - 2; i >= 0; i--)
    {
        temp = mul_mod(P, P, N);
        delete[] P.p;
        P = temp;
        if (x.p[i / 32] & (1 << (i % 32)))
        {
            temp = mul_mod(P, a, N);
            delete[] P.p;
            P = temp;
        }
    }
    return P;
}
  
```

// déclaration de P;
 //initialisation de la taille du tableau (32 éléments de 32 bits de longueur (unsigned int));
 //initialisation du tableau de unsigned int;

// size correspond au nombre d'élément dans le tableau(p) du lentier
 // pour toute la boucle for, copie du tableau de a dans P;

// On parcourt N en ordre descendant bit par bit;

// P = P*P reste N
 // si ei = 1

// on renvoie le résultat

Vecteurs de tests:

Quand il a fallu passer à la phase de test des vecteurs de test, nous nous sommes rendu compte que notre fonction ne retourne pas les résultats attendus. Même s'il n'est pas impossible que nous ayons fait une erreur dans notre algorithme, nous pensons que l'erreur vient de la fonction div_eucl dont le fonctionnement nous semble devenir aléatoire lorsque notre exposant x prends de grandes valeurs.

On obtient alors ces nombres en sortie :

```
[EN] C:\Users\loris\Google Drive (pinto.loris@hotmail.com)\JUT\INFO\Projet Info Groupe C\Debug\Projet Info Groupe C.exe
{888978359, 4220859661, 580528589, 838062124, 3311500355, 1385988532, 1840148337, 3973683792, 3433757303, 1588824920, 11^
58466016, 1495313493, 2394210304, 2716667977, 3833964921, 1079798547}
Appuyez sur une touche pour continuer... -
```

Alors que l'on est censé obtenir :

{3577248614, 1088852775, 2673576137, 1066220801, 592103025, 3455607712, 760091953, 3720316336, 3684356675, 2938667974, 2292421365, 2626684002, 3582190762, 704633395, 1964810293, 250614895}

Cependant pour des nombres relativement petits pour lesquels les résultats de test sont prévisibles, notre fonction semble fonctionner ce qui nous conforte dans l'idée que notre code est fonctionnel :

```
Que voulez-vous faire?
    1. Addition
    2. Soustraction
    3. Multiplication classique
    4. Division euclidienne (calcul du reste)
    5. Multiplication modulaire
    6. Exponentiation modulaire
Saisissez votre choix:6
-----
| Exponentiation modulaire d'entiers multiprécision: a^e mod N=
| avec a<N et e<N,
| sinon ils peuvent être réduits avant: a mod N (option 4 du menu) |
-----
Entrer le module (N=)
53
Entrer le premier opérande (a=)
52
Entrer l'exposant (e=)
48
a^b mod N=1
Souhaitez vous effectuer une autre opération? (o/n): -
```

On peut supposer que c'est dû au fonctionnement de la division euclidienne, l'algorithme exécuté n'étant pas le même si la diviseur est de taille 1 ou d'une taille supérieur à

VIII- Fonction dec2lentier

Présentation:

Le but de notre fonction est de convertir un entier de la base 10 vers la base 2^{32} , soit :

$$\sum_{k=0}^{n-1} b[i](10)^k \leftrightarrow \sum_{i=0}^{m-1} a[i](2^{32})^i$$

Nous avons donc recherché plusieurs méthodes pour passer d'une base $r_1 = 10$ à une base $r_2 = 2^{32}$.

Nous avons fini par retenir la méthode de Horner.

Méthode de Horner:

Définition mathématique:

Soit un polynôme défini de $\mathbb{R} \rightarrow \mathbb{R}$ par :

$$\sum_{i=0}^n a[i]X^i = a_n X^n + a_{n-1} X^{n-1} + \dots + a_2 X^2 + a_1 X^1 + a_0$$

Pour déterminer la valeur décimale d'un nombre en base r quelconque il suffit de calculer l'image par r du polynôme associé, par exemple :

$$(123)_{16} \Rightarrow r = 16$$

Son polynôme associé est : $1r^2 + 2r + 3 = 16^2 + 2 \cdot 16 + 3 = 256 + 32 + 3 = 291$

Alors, on a $(123)_{16} = (291)_{10}$

Application sur des bases r_1 et r_2 :

Cette méthode permet aussi de convertir une valeur de base r_1 vers une base r_2 :

Soit $r_1 = 10$ et $r_2 = 16$, en convertissant r_1 dans la base r_2 , nous savons que $(10)_{10} = (A)_{16}$
Prenons $(291)_{10}$, nous allons devoir convertir chacun de ses « chiffres » dans la base d'arrivée :

$$(2)_{10} = (2)_{16}; (9)_{10} = (9)_{16}; (1)_{10} = (1)_{16}$$

On peut alors écrire le polynôme associé en base r_2 :

$$\begin{aligned} (2 \cdot r_1^2 + 9 \cdot r_1 + 1)_{16} &= (2 \cdot A^2 + 9 \cdot A + 1)_{16} \\ &= (2 \cdot 64 + 9 \cdot A + 1)_{16} = (128 + 9 \cdot A + 1)_{16} = (122 + 1)_{16} = (123)_{16} \end{aligned}$$

Dans le cadre du projet:

Pour notre projet nous devions convertir un nombre de la base $r_1 = 10$ vers la base $r_2 = 2^{32}$

Pour ce faire nous avons donc repris la méthode de Horner, premièrement il est nécessaire de convertir $(r_1)_{10} = (r_1)_{2^{32}} = (10)_{2^{32}}$ car un « chiffre » en base $r_2 = 2^{32}$ peut prendre les valeurs comprises entre 0 et $2^{32} - 1$.

Ensuite, il faut convertir les « chiffres » du nombre en base 10 en « chiffres » en base 2^{32} , pour faire ceci en C++ nous avons créé une fonction `init_lentier`. Ici, nous noterons $\{a_n; a_{n-1}; \dots; a_1; a_0\}$ les « chiffres » convertis dans la base r_2 .

Puis il faut appliquer le même algorithme que précédemment :

$$\left(a_n \cdot r_1^n + a_{n-1} \cdot r_1^{n-1} + \dots + a_1 \cdot r_1 + a_0 \right)_{2^{32}}$$

On remarque alors que ceci revient à :

$$\left(\left(\left(\left(a_n \cdot r_1 \right) + a_{n-1} \right) r_1 + a_{n-2} \right) r_1 + \dots \right)_{2^{32}}$$

Ceci peut donc se traduire assez facilement en algorithmie, seulement cet algorithme utilise l'addition et la multiplication en base $r_2 = 2^{32}$, nous avons donc été contraint d'utiliser les fonctions `add_lentier` et `mult_classique`.

A) dec2lentier

Rôle : Convertir une chaîne de caractères représentant un entier en base 10, en une structure mémoire correspondant à un lentier. Ce qui équivaut à convertir un entier de la base 10 vers un entier en base $r = 2^{32}$.

Entrée : Un pointeur vers une chaîne de caractères.

Sortie : Un lentier, résultat de la conversion.

Nous présentons deux algorithmes pour la fonction `dec2lentier`, le premier est une version simplifiée illustrant le principe mathématique utilisé et le deuxième est utile à la traduction cette fonction en C++.

Déclaration de dec2lentier en C++:

```
/*
R: Convertir une chaine de caractères représentant un entier en base 10, en une
structure mémoire correspondant à un lentier. Ce qui équivaut à convertir un
entier de la base 10 vers un entier en base r=2^32.
E: 1 chaîne de caractères, nombre_dec.
S: 1 lentier, résultat de la conversion.
*/
lentier dec2lentier(char* nombre_dec);
```

Algorithme simplifié de dec2lentier:

```

dec2lentier : la fonction (nombre_dec : 1 pointeur vers 1 chaîne de caractère)
=> 1 lentier
{
    R: Convertir une chaîne de caractères représentant un entier en base 10,
    en une structure mémoire correspondant à un lentier. Ce qui équivaut à
    convertir un entier de la base 10 vers un entier en base r=2^32.
    E: 1 pointeur vers une chaîne de caractères, nombre_dec.
    S: 1 lentier, résultat de la conversion.
}

{ Lexique local }

buffer1, buffer2, digit, dix : 4 lentiers
idigit, nb_digit, i : 3 entiers non-signés

{ Algorithme local }

//Début
nb_digit <= taille_pratique_du_nombre_decimal
buffer1 <= 0 //on initialise le buffer1 à (0)2^32
dix      <= 10 //rl notre base de départ en base 2^32
i        <= 0

Tant que i < nb_digit

    idigit <= nombre_dec[i] //idigit est le "chiffre"
                           d'indice i en decimal

    digit <= idigit //convertit le idigit en base 2^32

    buffer2 <= buffer1 * dix //Multiplication

    buffer1 <= buffer2

    buffer2 <= buffer1 + digit //Addition

    buffer1 <= buffer2

FinTantque

Retourner buffer1
//Fin

```

Cet algorithme n'est pas viable pour la traduction en C++, il s'agit simplement de la version la plus « lisible » de celui-ci.

L'initialisation, la multiplication, l'addition et l'ajustement de la taille de lentier ne sont pas possible de cette manière, nous avons donc effectué un redécoupage fonctionnel pour notre fonction en utilisant les fonctions déjà créées par d'autres groupes comme `add_lentier`, `mult_classique` et `lAdjust_realloc`.

Pour l'initialisation d'un lentier nous avons créé une fonction supplémentaire `init_lentier`.

B) Fonction d'initialisation d'un lentier

Cette fonction permet donc d'initialiser un nombre en base $r = 2^{32}$ à partir d'un `unsigned int`. Cette fonction crée donc une structure mémoire contenant un lentier de taille pratique 1, et une case mémoire dans laquelle se trouve un entier en base 10 pouvant aller de 0 de à $2^{32} - 1$.

Déclaration de `Init_lentier` en C++:

```
/*
R : Initialise un lentier de taille 1 avec valeur comprise en 0 et (2^32)-1
E : Un entier
S : Un lentier
*/
lentier init_lentier(unsigned int a);
```

Algorithme de `Init_lentier`:

```
init_lentier : la fonction (a : 1 entier non-signé) => 1 lentier
    { Lexique local }
        n : 1 lentier

    { Algorithme local }

    //Debut
        n.size <= 1
        * (n.p) <= a

        Retourner n
    //Fin
```

Traduction en C++ de `Init_lentier`:

```
lentier init_lentier(unsigned int a) {
    lentier n;

    n.size = 1; //Le lentier ne contient qu'un seul digit
    n.p = new unsigned int;
    * (n.p) = a; //Le digit prend la valeur de l'entier passé en paramètre
    return n;
}
```

C) Fonction dec2lentier finale

Algorithme de dec2lentier:

```
dec2lentier : la fonction (nombre_dec : 1 pointeur vers 1 chaîne de caractère)
=> 1 lentier
{
    R: Convertir une chaîne de caractères représentant un entier en base 10,
    en une structure mémoire correspondant à un lentier. Ce qui équivaut à convertir
    un entier de la base 10 vers un entier en base r=2^32.
    E: 1 chaîne de caractères, nombre_dec.
    S: 1 lentier, résultat de la conversion.
}

{ Lexique local }

bufferl, buffer2, digit, dix : 4 lentiers
idigit, nb_digit, i : 3 entiers non-signés

{ Algorithme local }

//Début
nb_digit <= strlen(nombre_dec) //Taille de nombre_dec
bufferl <= init_lentier(0) //bufferl = (0) 2^32
dix <= init_lentier(10) //dix = (10) 2^32

i <= 0
Tant que i < nb_digit

    //Conversion forcée de char vers int
    idigit <= nombre_dec[i] - '0'

    //Convertit le idigit en lentier
    digit <= init_lentier(idigit)

    //multiplication des deux tampons
    buffer2 <= mult_classique(bufferl, dix)

    //Libération de la mémoire occupée par le pointeur
    Effacer bufferl.p

    bufferl <= buffer2

    //On ajoute la valeur du digit
    buffer2 <= add_lentier(bufferl, digit)

    //Libération de la mémoire occupée par le pointeur
    Effacer bufferl.p
    Effacer digit.p

    bufferl <= buffer2

    //On Réajuste la taille de lentier
    lAdjust_realloc(bufferl)

FinTantque

//Libération de la mémoire occupée par le pointeur
Effacer dix.p

Retourner bufferl
//Fin
```

Traduction en C++ de dec2lentier:

```
lentier dec2lentier(char* nombre_dec) {
    //Structures lentier nécessaires
    lentier buffer1, buffer2, digit, dix;

    //Nombre 10^n en base 10
    unsigned int idigit;

    //Taille de la chaîne de caractères
    unsigned int nb_digit = strlen(nombre_dec);

    //Initialisation buffer1 = 0 et dix = 10 (en base 2^32)
    buffer1 = init_lentier(0);
    dix = init_lentier(10);

    //L'algorithme qui suit reprend la méthode de Horner
    //
    for (unsigned int i = 0; i < nb_digit; i++) {
        idigit = nombre_dec[i] - '0'; //Nombre d'indice i dans la chaîne
        (Conversion forcé de type)

        digit = init_lentier(idigit); //Conversion du digit (base 10)-->(base 2^32)

        //Multiplication par 10 du buffer2
        buffer2 = mult_classique(buffer1, dix);

        delete[] buffer1.p;

        buffer1 = buffer2; //Swap buffer1 et buffer2

        //Ajout du au buffer2 du digit
        buffer2 = add_lentier(buffer1, digit);

        delete[] buffer1.p;
        delete[] digit.p;

        buffer1 = buffer2; //Swap buffer1 et buffer2

        lAdjust_realloc(buffer1);
    }

    //Libération de la mémoire avant la fin de la fonction
    delete[] dix.p;

    return buffer1;
}
```

strlen : Renvoie la longueur de la chaîne d'octets donnée, c'est-à-dire le nombre de caractères dans un tableau de caractères dont le premier élément est pointé par str jusqu'au premier caractère nul inclus. Le comportement n'est pas défini s'il n'y a pas de caractère nul dans le tableau de caractères pointé par str. C'est une fonction de la bibliothèque et du module std iostream.

Vecteurs de tests:

Pour tester notre fonction nous avons créé une fonction `main` dans laquelle tous les vecteurs de tests qui nous ont été fournis sont répertoriés et donc testés.

Main

```
#include <iostream>
#include "Header_Dec2lentier.h"

using namespace std;

int main()
{
    /* le jeu de données */
    //début copier/coller (dans votre main)
    char * s = (char
*)"20137450181711572238169339904695315491098153870297777299155983749081949482950
8978805309598983915928634998";
    lentier a;
    a = dec2lentier(s);
    Affiche_lentier(a);//fin copier/coller
    /*
    Résultat attendu:
    {2045593206,279951739,222109666,889033077,1057640747,1016240163,1712087700
,1631452143,173372631,1544088762,94277024};
    */
    //Pensez à la libération mémoire avant de faire un autre test
    delete[] a.p;
    /****** fin test 1 ******/

    /* 2e jeu de données */
    //début copier/coller (dans votre main)
    s = (char
*)"91649930033805470816987702210767603581554787441700819666469792157668448721505
866052556277680677740731818648996689113101990157422513524";
    a = dec2lentier(s);
    Affiche_lentier(a);//fin copier/coller
    /*
    Résultat attendu:
    {864374132,1058999259,417803488,368991873,1147633573,1501943628,1410920752
,1717868749,1020262389,1835849410,14035462,1469095785,1576932299,541569180};
    */
    //Pensez à la libération mémoire avant de faire un autre test
    delete[] a.p;
    /****** fin test 2 ******/

    /* 3e jeu de données */
    //début copier/coller (dans votre main)
    s = (char *)"346717102085642256956601510400750434070696828726";
    a = dec2lentier(s);
    Affiche_lentier(a);//fin copier/coller
    /*
    Résultat attendu:
    {456924982,1120800428,1930852920,377386224,1018909987};
    */
    //Pensez à la libération mémoire avant de faire un autre test
    delete[] a.p;
    /****** fin test 3 ******/
```

```
/* 4e jeu de données */
    //début copier/coller (dans votre main)
    s = (char *) "128011798607918111696488314221889295570";
    a = dec2lentier(s);
    Affiche_lentier(a); //fin copier/coller
/*
Résultat attendu:
{599495890,1907943064,1120961758,1615736053};
*/
//Pensez à la libération mémoire avant de faire un autre test
delete[] a.p;
***** fin test 4 *****

/* 5e jeu de données */
//début copier/coller (dans votre main)
s = (char *)
*) "12953077114869672789688901852203286587466359754925709235712200219984374305155
284430439174025265561522645360443017174514041037";
    a = dec2lentier(s);
    Affiche_lentier(a); //fin copier/coller
/*
Résultat attendu:
{604930253,1789108684,869152230,699207277,1021885146,1733526362,1758206536
,1439688634,2102518236,758356462,794148614,1365955340,328741563};
*/
//Pensez à la libération mémoire avant de faire un autre test
delete[] a.p;
***** fin test 5 *****

/* 6e jeu de données */
//début copier/coller (dans votre main)
s = (char *)
*) "10838118778516132338152572683238206397964512774422399776816437333735747579608
119866212445811702147183211705527675342";
    a = dec2lentier(s);
    Affiche_lentier(a); //fin copier/coller
/*
Résultat attendu:
{1054321102,342777026,1136023140,483769753,884346206,1334705544,940694735,
2005146635,1118074816,1318080959,876572974,1181395827};
*/
//Pensez à la libération mémoire avant de faire un autre test
delete[] a.p;
***** fin test 6 *****

/* 7e jeu de données */
//début copier/coller (dans votre main)
s = (char *) "665178495054796742";
a = dec2lentier(s);
Affiche_lentier(a); //fin copier/coller
/*
Résultat attendu:
{637032390,154873937};
*/
//Pensez à la libération mémoire avant de faire un autre test
delete[] a.p;
***** fin test 7 *****
```

Ceci allant du vecteur de test n°1 au n°20, nous ne mettons pas l'entièreté du main car ce serait fastidieux.

Le résultat d'un entier passé en paramètre est affiché par la fonction Affiche_lentier

Exécution du programme et résultats sur la console de test

Après exécution de la fonction main la console de débogage affiche ceci :

```
Console de débogage Microsoft Visual Studio
{2045593206,279951739,222109666,889033077,1057640747,1016240163,1712087700,1631452143,173372631,1544088762,94277024} Vecteur de test n°1
{864374132,1058999259,417803488,368991873,1147633573,1501943628,1410920752,1717868749,1020262389,1835849410,14035462,1469095785,1576932299,541569180} n°2
{456924982,1120800428,1930852920,377386224,1018909987} n°3
{599495890,1907943064,1120961758,1615736053} n°4
{604930253,1789108684,869152230,699207277,1021885146,1733526362,1758206536,1439688634,2102518236,758356462,794148614,1365955340,328741563} n°5
{1054321102,342777026,1136023140,483769753,884346286,1334705544,940694735,2005146635,1118074816,1318880959,876572974,1181395827} n°6
{637032390,154873937} n°...
{2109579506,759804190,1027454291,831248089,1459011467,2049339437,417290803,1069734356}
{372325391,1828090818,2135693837,1738280731,9348733,1802620392,645118186,352125759}
{1128887939,1236471966,2125865428,2069582675,1094134953,1096456596,1240179986,19780707927,130368775,1010273188,468256669,285242712,248618795}
{1045046902,1276073086,1253600616,356574722,1177928876,16708891420,1426309078,371989651,2043216811,1106916248,360199041,1634013895,1116264981,15335785,131648433,1468390741}
{1260536372,557379059,784877450,1182635399,1651514012,1881334047,275331738}
{2011702822,1285604926,1934994960,149461887}
{209863839,1194508789,662813159,1463464456,1551083511,1840742035,986872228,829908941,65248039,882605391}
{425447080,369135638,905606523,440782805,500784071,226513616}
{1761320444,783892675,2032155986,796472195,287923039,1766006385,1071803933,1762661330}
{209925211,1550172642,1779687446,1744148932,1760036481,826712588,259478444,1076017289}
{2100220479,2062889517,1060221393,17984870}
{849562934,443431950,1167146899,1755169457,884214816,1667930971,1981683073,2131493351,1281767767,618092100,2016165689,2078239962,906015139,1634688426}
{521192821,1117430338,1212485459,2071365463,749634136,809150744,1683918297,1576346724,1068629188}

Sortie de C:\Users\matteo\OneDrive\Documents\VC++\Arithmetique_multiprecision\Debug\Arithmetique_multiprecision.exe (processus 15228) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.
```

Ainsi chaque ligne correspond à un vecteur de test de 1 à 20.

On s'aperçoit ainsi que les entiers renvoyés par notre fonction `dec21entier` correspondent bien aux valeurs fournies dans les vecteurs de tests. Nous aurions pu réaliser une fonction permettant de renvoyer « réussi » si le résultat renvoyé par `dec21entier` correspondait bien au vecteur de test ou « échoué » dans le cas contraire.

Cependant, nous avons testé notre fonction avec la fonction `lentier2dec` créée par le groupe de Sauze E., Adam M., Pinto L.

Nous avons donc testé si en passant une chaîne de caractères, initialisée au préalable, à `dec21entier` puis en faisant passer le résultat renvoyé à `lentier2dec` la fonction renvoyait bien la même chaîne de caractère qu'en entrée, et c'est bien le cas.

Ci-dessous un des mains que nous avions testé en utilisant les deux fonctions.

Main test de dec2lentier et lentier2dec

```
int main() {
    //Lexique Principal
    char * s = (char *)"1234567891011121314151617"

    lentier Lent;

    //Algorithme Principal
    //Debut
    Lent = dec2lentier(lent)
    /*
        Lent.size = 3;
        Lent.p = new.unsigned int[3];
        Lent.p[0] = 3197516993;
        Lent.p[1] = 255446423;
        Lent.p[2] = 66926;
    */
    char* dec = new char[25];
    dec = lentier2dec(Lent);

    for (int i=0; i < 25; i++) {
        cout << dec[i];
    }
    cout << endl << endl;

    //Fin

    delete[] Lent.p, dec;

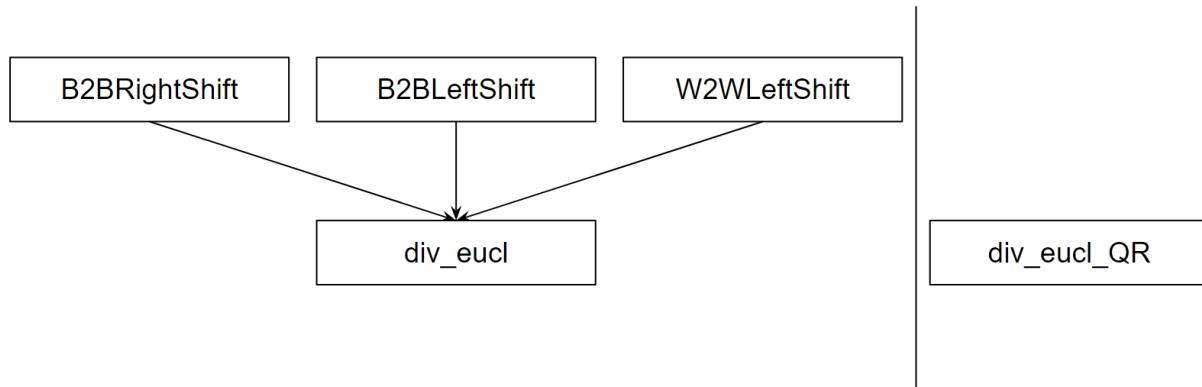
    system("pause")

    return 0;
}
```

La fonction `lentier2dec` renvoyait bien `1234567891011121314151617`.

IX- Fonction div_eucl

Re-découpage fonctionnel :



Nous avons décidé de créer 3 sous-fonctions pour faciliter le travail. En effet, les décalages à gauche et à droite, bit à bit ou mot à mot sont des tâches répétitives (boucles) qui nous servent plusieurs fois dans notre fonction div_eucl, il est donc pertinent d'en faire des fonctions à part entière et de les appeler quand nécessaire:

A) B2BRightShift

Cette fonction va permettre de décaler, bit à bit, à droite, un lentier d'un certain unsigned int "amount" renseigné en paramètre. Ainsi, nous aurons trois paramètre d'entrée, un lentier a, un unsigned int "amount" qui correspond au nombre de fois que l'on veut décaler à droite le lentier et enfin un troisième paramètre (char) : "modifySize" si ce dernier est égale à 1, la fonction va appeler la fonction "lAdjust_realloc" qui va permettre d'adapter la taille du lentier à la taille pratique.

Rôle : décalage bit à bit à droite de amount d'un lentier a , si modifySize = 1, alors la taille du lentier va être adaptée à la taille pratique. Amount ≤ 32 .

Entrée : 1 lentier a, int amount, char modifySize (1 ou 0, automatiquement à zero)

Sortie : lentier

Déclaration de fonction :

B2BRightShift : la fonction (a : 1 lentier, amount : 1 entier, modifySize : 1 caractère) -> 1 lentier

Lexique local :

buffer, i	: 2 entiers non signés
result	: 1 lentier

Algorithme local :

Début

```
    buffer <- 0
    result.size <- a.size
    result.p <- Allouer(result.size, entiers non signés)
    {Chaque case de result est initialisées à 0}
    Pour i de a.size à 1
        result.p[i - 1] <- (a.p[i - 1] >> amount) + buffer
        buffer <- a.p[i] << (32 - amount)
    FinPour
    Si modifySize = 1
        Alors
            lAdjust_realloc(result)
    FinSi
    Retourner result
Fin
```

B) B2BLeftShift

Cette fonction va permettre de décaler, bit à bit, à gauche, lentier d'un certain unsigned int "amount" renseigné en paramètre. Tout comme B2BRightShift, elle possède aussi un troisième paramètre d'entrée : "modifySize", si ce dernier est égale à 1, la fonction va alors augmenter la taille du lentier, si nécessaire, pour éviter la perte de données en cas de débordement.

Rôle : décalage bit à bit à gauche de amount d'un lentier a , si modifySize = 1, alors la taille du lentier va être augmentée Amount \leq 32. pour éviter la perte de donnée par débordement.

Entrée : 1 lentier a, int amount, char modifySize (1 ou 0, automatiquement à zero)

Sortie : lentier

Déclaration de fonction :

B2BLeftShift : la fonction (a : 1 lentier, amount : 1 entier, modifySize : 1 caractère) \rightarrow 1 lentier

Lexique local :

buffer, i	: 2 entiers non signés
result	: 1 lentier

Algorithme local :

Début

```
    buffer <- 0
    result.size <- a.size + 1
    result.p <- Allouer(result.size, entiers non signés)
        {Chaque case de result est initialisées à 0}
    Pour i de 0 à a.size - 1
        result.p[i] <- (a.p[i] << amount) + buffer
        buffer <- (a.p[i] >> (32 - amount))
    FinPour
    Si modifySize = 1
        Alors
            result.p[a.size] <- buffer
    FinSi
    lAdjust_realloc(result)
    Retourner result
Fin
```

C) W2WLeftShift

Cette fonction va permettre de décaler, mot à mot, à gauche, lentier d'un certain unsigned int "amount" renseigné en paramètre, la taille augmentera automatiquement. Cette fonction va notamment servir lors des exponentiations (e.g. base^32)

Rôle : décalage mot à mot à gauche de amount d'un lentier a, la taille augmente automatiquement

Entrée : 1 lentier a, int amount

Sortie : lentier

Déclaration de fonction :

W2WLeftShift : la fonction (a : 1 lentier, amount : 1 entier) -> 1 lentier

Lexique local :

result : 1 lentier
i : 1 entier non signé

Algorithme local :

Début

```
    result.size <- a.size + amount
    result.p <- Allouer(result.size, entiers non signés)
    {Chaque case de result est initialisées à 0}
    Pour i de 1 à a.size
        result.p[result.size - i] <- a.p[a.size - i]
    FinPour
    Retourner result
```

Fin

D) div_eucl_QR

div_eucl_QR est une fonction spéciale qui retourne le quotient ET le reste que nous avons créé à la demande du groupe en charge de la fonction, pour ce faire, elle utilise le type composé quores, créé par le groupe de lentier2dec. Elle permet seulement la division par un lentier b d'une longueur de 1 mot.

Déclaration de fonction :

div_eucl_QR : la fonction (a : 1 lentier, b : 1 lentier) -> 1 quores

Lexique local :

i : 1 entier
temp : 1 entier non signé sur 64 bits
res : 1 quores

Algorithme local :

Début

```
temp <- 0
res.quotient.size <- a.size

res.quotient.p <- new unsigned int[res.quotient.size]
```

```
Pour i de a.size à 1
temp <- (temp << 32) + a.p[i - 1]
res.quotient.p[i - 1] <- temp div b.p[0]
temp <- temp reste b.p[0]
FinPour
lAdjust(res.quotient)
res.reste <- temp
Retourner res
```

Fin

E) div_eucl

La fonction principale div_eucl permet de diviser par un b d'une longueur ≥ 1 .

Rôle : Divise les lentiers a par b et retourne le reste

Entrée : 2 lentiers a et b ($b \neq 0$)

Sortie : 1 lentier résultat de $a \% b$

Déclaration de fonction :

div_eucl : la fonction (a : 1 lentier, b : 1 lentier) -> 1 lentier

Lexique local :

i	: entier non signé {i est un compteur}
lambda	: caractère non signé
BASE	: entier non signé constant sur 64 bits
tempLL	: entier non signé sur 64 bits
q, r, buffer1, buffer2, buffer3, buffer4, na, nb : 9 lentier	
{q et r représentent respectivement le quotient et le reste}	

Algorithme local :

```

Début
    BASE <- 0x100000000
    Si b.size = 1
        Alors
            tempLL <- 0
            Si a.size = 1
                Alors
                    q.size <- 1
                Sinon
                    q.size <- a.size - 1
                FinSi
                q.p <- Allouer(q.size, entier non signé)
                Pour i de a.size à 1
                    tempLL <- (tempLL << 32) + a.p[i - 1]
                    q.p[i - 1] <- tempLL div b.p[0]
                    tempLL <- tempLL reste b.p[0];
                FinPour
                r.size <- 1
                r.p <- Allouer(1, entier non signé)
                r.p[0] <- tempLL
            Sinon
                Si a.size = b.size
                    Alors
                        q.size <- 1
                    Sinon
                        q.size <- a.size - b.size
                    FinSi
                    na.size <- a.size
                    na.p <- Allouer(na.size, entier non signé)
                    i <- 0
                FinSi
            FinSi
        FinAlors
    FinSi
FinDébut

```

```

Tant que i < a.size
    na.p[i] <- a.p[i]
    i <- i +1
FinTantQue

nb.size <- b.size
nb.p <- Allouer(nb.size, entier non signé)
Pour i de 0 à b.size
    nb.p[i] <- b.p[i]
FinPour

Pour i de 0 à n - t
    Qj <- 0
FinPour
{
Cette partie n'a pas été mise dans le code
car chaque mots du quotient sont initialisés à 0 quand q est déclaré
}

Si cmp_lentier(na, nb) = -1
    Alors
        {
            la fiche technique impose que a.size et b.size soient égaux
            ici nous vérifions si a et b sont égaux
        }
        q.size <- 1
        Effacer q.p
        q.p <- Allouer(1, entier non signé)
        q.p[0] <- 0
        {
            Cette partie n'est nécessaire que si nous renvoyons le quotient
        }
    Sinon
        {Optimisation "lambda"}
        lambda <- 0
        Tant que (nb.p[nb.size - 1] < (BASE / 2))
            buffer1 <- B2BLeftShift(na, 1, 1)
            Effacer na.p
            na <- buffer1

            buffer1 <- B2BLeftShift(nb, 1, 1)
            Effacer nb.p
            nb <- buffer1

            lambda <- lambda + 1
        FinTantQue

        Si na.size > nb.size
            Alors
                buffer1 <- W2WLeftShift(nb, na.size - nb.size)
            Sinon
                buffer1 <- nb
        FinSi
        {
Ici, buffer1 représente B multiplié à la puissance (a.size - b.size)
on ne fait pas le calcul si nb.size = na.size pour gagner en optimisation
        }

        Tant que (cmp_lentier(na, buffer1) ≥ 0)
            q.p[na.size - nb.size] <- q.p[na.size - nb.size] + 1
            buffer2 <- sub_lentier(na, buffer1)
            Effacer na.p
            na <- buffer2
        FinTantQue

        Pour i de a.size - 1 à b.size
            Si na.p[i] = nb.p[nb.size - 1]
                Alors
                    q.p[i - nb.size] <- BASE - 1
                Sinon
                    temp1 <- (na.p[i] << 32) + na.p[i - 1]
                    q.p[i - nb.size] <- temp1 div nb.p[nb.size - 1]
            }
}

```

```

FinSi

    buffer1.p <- Allouer(3, entier non signé)
    buffer1.size <- 3
    buffer1.p[2] <- na.p[i]
    buffer1.p[1] <- na.p[i - 1]
    buffer1.p[0] <- na.p[i - 2]

    buffer2.p <- Allouer(2, entier non signé)
    buffer2.size <- 2
    buffer2.p[1] <- nb.p[nb.size - 1]
    buffer2.p[0] <- nb.p[nb.size - 2]

    buffer3.p <- Allouer(1, entier non signé)
    buffer3.size <- 1
    buffer3.p[0] <- q.p[i - nb.size]

    lAdjust_realloc(buffer1)
    lAdjust_realloc(buffer2)

    buffer4 <- mult_classique(buffer2, buffer3)

    Tant que cmp_lentier(buffer4, buffer1) = 1
        q.p[i - nb.size] <- q.p[i - nb.size] - 1
        buffer3.p[0] <- q.p[i - nb.size]

        Effacer buffer4.p
        buffer4 <- mult_classique(buffer2, buffer3)
    FinTantQue

    Effacer buffer1.p
    Effacer buffer2.p
    Effacer buffer3.p
    Effacer buffer4.p

    buffer1.p <- Allouer(1, entier non signé)
    buffer1.size <- 1
    buffer1.p[0] <- q.p[i - nb.size]
    buffer2 <- W2WLeftShift(buffer1, i - nb.size)
    Effacer buffer1.p

    buffer1 <- mult_classique(buffer2, nb)
    Effacer buffer2.p
    Si cmp_lentier(na, buffer1) = -1
        Alors
            q.p[i - nb.size] <- q.p[i - nb.size] - 1
            Effacer buffer1.p
            buffer1.p <- Allouer(1, entier non signé)
            buffer1.size <- 1
            buffer1.p[0] <- q.p[i - nb.size]
            buffer2 <- W2WLeftShift(buffer1, i - nb.size)
            Effacer buffer1.p
            buffer1 <- mult_classique(buffer2, nb)
            Effacer buffer2.p
        FinSi
        buffer2 <- sub_lentier(na, buffer1);
        Effacer na.p;
        Effacer buffer1.p
        na <- buffer2
    FinPour
FinSi
Si lambda > 0
    Alors
        buffer1 <- B2BRightShift(na, (int)lambda, 1)
        Effacer na.p
        na <- buffer1
    FinSi
    r <- na
    Effacer nb.p
    Effacer q.p
    lAdjust_realloc(r)
FinSi
Retourner r
Fin

```

Ici, les cases des pointeurs sont indiqués avec des crochets car notre éditeur de texte ne me permettait pas de mettre du texte en indice et vu qu'il y avait beaucoup d'indentations, on ne pouvait pas non plus mettre le texte brut, sinon il y avait des retours à la ligne.

Implémentation en C :

```

406 lentier div_eucl(lentier a, lentier b) {
407     // Variables locales
408     unsigned int i; // compteur
409     unsigned char lambda;
410     const unsigned long long int BASE = 0x100000000;
411     unsigned long long int templl;
412     lentier q, r, buffer1, buffer2, buffer3, buffer4, na, nb;
413     //q = quotient, r = reste
414
415     // Algorithme
416     if (b.size == 1) {
417         templl = 0;
418         q.size = (a.size == 1) ? (1) : (a.size - 1);           //q.size ne peut pas être égale à zero
419         q.p = new unsigned int[q.size];
420
421         for (i = a.size; i > 0; i--) {
422             templl = (templl << 32) + a.p[i - 1];           //On fait la division euclidien mot
423             par mot, le reste sera ajouter au mot suivant (d'où le long long int) et ainsi de suite jusqu'au
424             reste final
425             q.p[i - 1] = (unsigned int)(templl / b.p[0]);
426             templl = templl % b.p[0];
427         }
428         r.size = 1;
429         r.p = new unsigned int[1];
430         r.p[0] = (unsigned int)templl;
431     }
432     else {
433         q.size = (a.size == b.size) ? (1) : (a.size - b.size);
434         q.p = new unsigned int[q.size]();
435
436         // na.p = a.p; on fait des copie dans des nouveaux espaces mémoire car nous allons travailler
437         // directement sur na et nb, sans modifier a et b
438         na.size = a.size;
439         na.p = new unsigned int[na.size];
440         for (i = 0; i < a.size; i++) {
441             *(na.p + i) = *(a.p + i);
442         }
443
444         //nb.p = b.p;
445         nb.size = b.size;
446         nb.p = new unsigned int[nb.size];
447         for (i = 0; i < b.size; i++) {
448             nb.p[i] = b.p[i];
449         }
450     }
451 }
```

```
446     }
447
448     /*
449     Partie 1 :
450     Pas besoin de le faire, les bits de q ont été initialisé à 0 lors de sa déclaration avec les
451     parenthèses après les [] (voir page 2 fascicule de projet)
452     */
453     if (cmp_lentier(na, nb) == -1) {
454         //il est demandé que A et B aient au moins le même nombre de mots mais il n'est pas dit que A doit
455         //être supérieur à B
456
457         /*
458         q.size = 1;
459         delete q.p[];
460         q.p = new unsigned int[1]();
461         N'est nécessaire que si nous devons retourner le quotient
462         */
463     }
464     else {//Algo donné
465         //Optimisation "lambda" On multiplie na et nb par 2 tant que le mot de poids fort de nb <= base/2
466         lambda = 0;
467         while (nb.p[nb.size - 1] < BASE / 2) {
468
469             buffer1 = B2BLeftShift(nb, 1, 1);
470             delete[] nb.p;      //On libère l'espace
471             nb = buffer1;
472
473             buffer1 = B2BLeftShift(nb, 1, 1);
474             delete[] nb.p;      //On libère l'espace
475             nb = buffer1;
476
477             ++lambda;
478         }
479
480         // Partie 2 :
481         if (na.size > nb.size) {
482             // ici le Buffer1 correspond à B multiplié par la base à la puissance n-t (équivalent à a.size
483             // - b.size)
484             buffer1 = W2WLeftShift(nb, na.size - nb.size);
485             //Multiplier par la base revient à décaler d'un mot à gauche, et pas la base puissance x, de x
```

```
485     mots à gauche
486 }
487 else {
488     buffer1 = nb;
489 }
490 while (cmp_lentier(na, buffer1) >= 0) { //Tans que na >= nb, on ajoute 1 au quotient et on
491     soustrait nb à na
492     q.p[na.size - nb.size] = q.p[na.size - nb.size] + 1;
493     buffer2 = sub_lentier(na, buffer1);
494     delete[] na.p;      //On libère l'espace
495     na = buffer2;
496 }
497
498 // Partie 3 :
499 for (i = a.size - 1; i >= b.size; --i) {
500     // 3.a)
501     if (na.p[i] == nb.p[nb.size - 1]) {
502         q.p[i - nb.size] = BASE - 1;
503     }
504     else {
505         temp1l = (((unsigned long long int)na.p[i]) << 32) + na.p[i - 1];
506         q.p[i - nb.size] = ((unsigned int)(temp1l / nb.p[nb.size - 1]));
507     }
508
509     // 3.b)
510
511     //Création des 3 buffers nécessaires
512     buffer1.p = new unsigned int[3];
513     buffer1.size = 3;
514     buffer1.p[2] = na.p[i];
515     buffer1.p[1] = na.p[i - 1];
516     buffer1.p[0] = na.p[i - 2];
517
518     buffer2.p = new unsigned int[2];
519     buffer2.size = 2;
520     buffer2.p[1] = nb.p[nb.size - 1];
521     buffer2.p[0] = nb.p[nb.size - 2];
522
523     buffer3.p = new unsigned int[1];
524     buffer3.size = 1;
525     buffer3.p[0] = q.p[i - nb.size];
```

```

525 //Ajustement des tailles pour optimiser les multiplications
526 lAdjust_realloc(buffer1);
527 lAdjust_realloc(buffer2);
528
529 buffer4 = mult_classique(buffer2, buffer3);
530
531 while (cmp_lentier(buffer4, buffer1) == 1) {
532     q.p[i - nb.size] = q.p[i - nb.size] - 1;
533     buffer3.p[0] = q.p[i - nb.size];
534
535     delete[] buffer4.p;
536     buffer4 = mult_classique(buffer2, buffer3);
537 }
538 delete[] buffer1.p; //On libère l'espace
539 delete[] buffer2.p;
540 delete[] buffer3.p;
541 delete[] buffer4.p;
542
543 // 3.c) et 3.d)
544 buffer1.p = new unsigned int[1];
545 buffer1.size = 1;
546 buffer1.p[0] = q.p[i - nb.size];
547 buffer2 = W2WLeftShift(buffer1, i - nb.size); //Q[i-t]*base^(i-t)
548 delete[] buffer1.p; //On libère l'espace
549
550 buffer1 = mult_classique(buffer2, nb); //B*Q[i-t]*base^(i-t)
551 delete[] buffer2.p; //On libère l'espace
552
553 if (cmp_lentier(nb, buffer1) == -1) { //Si A < buffer1, alors la soustraction A - B*Q
554 [i-t]*base^(i-t) donnera un résultat négatif, or nous travaillons avec des entiers non-signés,
555 il faut donc d'abord s'assurer que A >= buffer1 pour pouvoir effectuer l'opération
556     q.p[i - nb.size] = q.p[i - nb.size] - 1; //On soustrait 1 à Q[i-t] puis on refait les
557     mêmes étapes
558
559     delete[] buffer1.p; //On libère l'espace
560
561     buffer1.p = new unsigned int[1];
562     buffer1.size = 1;
563     buffer1.p[0] = q.p[i - nb.size];
564     buffer2 = W2WLeftShift(buffer1, i - nb.size);
565     delete[] buffer1.p; //On libère l'espace
566
567     buffer1 = mult_classique(buffer2, nb);
568     delete[] buffer2.p; //On libère l'espace
569
570 //Puis on effectue la soustraction
571 buffer2 = sub_lentier(nb, buffer1);
572 delete[] nb.p; //On libère l'espace
573 delete[] buffer1.p; //On libère l'espace
574 na = buffer2;
575
576 }
577 //Maintenant il faut diviser le reste par lamda (si l'optimisation a eu lieu)
578 if (lambda > 0) {
579     buffer1 = B2BRightShift(nb, (int)lambda, 1);
580     delete[] nb.p; //On libère l'espace
581     na = buffer1;
582 }
583 r = na; //R prend la valeur restante de na
584 delete[] nb.p; //On libère l'espace
585 delete[] q.p;
586 lAdjust_realloc(r); //On ajuste la taille de r avant de la retourner
587
588 return r;
589 }
590

```

Vecteurs de tests;

```

6   Int main() {
7       //1er test
8
9       unsigned int a[22] = { 293034349,844693298,1814842293,1421314185,1691207896,638289000,1203477101,142092880,130274739,1610018115,1744525153,533928273,677033824,1257
10      unsigned int b[13] = { 1125228701,2090755820,864880040,89957166,1761468921,720997415,10652816,594182948,1691339135,303687165,1438876247,1358697781,3872484998 };
11      lentier aa;
12      aa.p = a;
13      aa.size = 22;
14      lentier bb;
15      bb.p = b;
16      bb.size = 13;
17      lentier res;
18      res = div_eucl(aa, bb);
19      Affiche_lentier(res);
20      delete [] res.p;
21
22      /* Résultat attendu: {1701055259,2437423851,1634014391,3166386563,3335268908,2490487792,3435823235,429906975,426599418,1820046964,3402462427,47390074,1653958660}; */
23
24
25
26 // 2eme test
27 unsigned int a[31]={789994883,1124693375,1151577872,243529271,721734881,1685586166,1120563695,1979283156,1874126978,1876742024,1928414452,1917691205,33886726,862526042,777512624,1987199350,1987754743,728784796,784595751,2877711989,334770670,1425593166,2088364726,
28 unsigned int b[17]={683880749,180155848,181745792,846610629,9018598721,719766290,1967173115,733950229,448489613,896411492,514621833,216817178,938118218,1577147875,994129794,7790833929,3364011831};
29      lnter aa;
30      aa.p=a;
31      aa.size=31;
32      lnter bb;
33      bb.p=b;
34      bb.size=17;
35      lentier res;
36      res=div_eucl(aa,bb);
37      Affiche_lentier(res);
38      delete [] res.p;
39
40      /* Résultat attendu: {1864164160,1326969350,805337115,2408431157,1471102048,3970883249,940133982,3806410607,3897992605,2918391274,3008908667,852014235,2306929951,1080868555,680612480,3139900885,3262575601}; */
41
42
43
44 // 3eme test
45 unsigned int a[18]={1147646432,2049684661,752739199,1088527510,831154031,1722187853,1333095753,1051409649,1982850640,1155181699,186962114,1933016560,1758262449,287117954,967276704,457388821,1189008676,1687042995};
46 unsigned int b[17]={177078289,1922698905,2113452608,1173509781,289836291,261586138,2183827999,1666983366,1196715924,726378281,736917889,764165987,53524311,1884564321,665347880,806263512,2973091831};
47      lnter aa;
48      aa.p=a;
49      aa.size=18;
50      lnter bb;
51      bb.p=b;
52      bb.size=17;
53      lentier res;
54      res=div_eucl(aa,bb);
55      Affiche_lentier(res);
56      delete [] res.p;
57
58      /* Résultat attendu: {1267307812,2112174816,1724038913,3212396018,3192663910,3417155958,3662038052,1253982747,2070720983,4146489717,3471551173,1508031269,1586699791,3634978718,1402790920,2696397809,1032395073}; */
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
79

```

Citise 1 - GROUPE TD C

compte rendu final

```
// 4ème test

unsigned int a[16]={11220288,401517913,136334709,1166401988,508480027,2069351269,777180789,795597981,889144326,1234569610,1984606657,428703673,1511647899,1759821915,414672633,537674832};
unsigned int b[10]={2049658286,617258763,494010304,1569157924,1813974687,1220396665,158592165,430637907,1273920978,4198640134};
lentier aa;
aa.p-a;
aa.size=16;
lentier bb;
bb.p-b;
bb.size=10;
lentier res;
res=div_eucl(aa,bb);
Affiche_lentier(res);
delete []res.p;

/* Résultat attendu: {781057304,434514759,2785139496,1398427867,560981540,466983461,1761508413,3878945907,518227962,1476513437}; */

Console de débogage Microsoft Visual Studio
{781057304,434514759,2785139496,1398427867,560981540,466983461,1761508413,3878945907,518227962,1476513437}

Sortie de C:\tmp\info1\projet\projet_division_vf\projet_info\Debug\projet_info.exe (processus 74856) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.

la sortie à partir de: Débrouer
```



```
//Séme test

unsigned int a[25]={721281021,447004051,313668568,732501310,848521964,450003270,1898503258,1357001991,371870851,528660439,5116325,1261015217,1763170049,1989722582,1689718590,1127334301,1602061249,2104391523,16650083
unsigned int b[13]={1084502249,671295585,358277965,980175088,1768281372,291077908,1701456109,67801776,694746468,286473771,916323740,1954749738,2185377069};
lentier aa;
aa.p-a;
aa.size=25;
lentier bb;
bb.p-b;
bb.size=13;
lentier res;
res=div_eucl(aa,bb);
Affiche_lentier(res);
delete []res.p;

/* Résultat attendu: {1520033227,2617285412,4027055362,90859088,1900236232,1779791506,741012039,2782373289,29061390,486035685,3117372191,1916659523,1618572242}; */

Console de débogage Microsoft Visual Studio
{1520033227,2617285412,4027055362,90859088,1900236232,1779791506,741012039,2782373289,29061390,486035685,3117372191,1916659523,1618572242}

Sortie de C:\tmp\info1\projet\projet_division_vf\projet_info\Debug\projet_info.exe (processus 57876) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.

la sortie à partir de: Débrouer
```



```
// Résultat attendu: {1520033227,2617285412,4027055362,90859088,1900236232,1779791506,741012039,2782373289,29061390,486035685,3117372191,1916659523,1618572242}; */

// 5ème test

unsigned int a[13]={566493869,130658409,540151199,18120062,2120681391,82387442,1309514563,1575258993,39295317,827039248,932011152,613461956,838582318};
unsigned int b[6]={1857921236,854119634,2070522852,754535838,1525415219,242889117};
lentier aa;
aa.p-a;
aa.size=13;
lentier bb;
bb.p-b;
bb.size=6;
lentier res;
res=div_eucl(aa,bb);
Affiche_lentier(res);
delete []res.p;

/* Résultat attendu: {886481472,9334121,3752685010,2049342436,783513041,2302941576}; */

Console de débogage Microsoft Visual Studio
{886481472,9334121,3752685010,2049342436,783513041,2302941576}

Sortie de C:\tmp\info1\projet\projet_division_vf\projet_info\Debug\projet_info.exe (processus 19036) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.
```

Citise 1 - GROUPE TD C

compte rendu final

Citise 1 - GROUPE TD C

compte rendu final

```
257 // 11 ème test
258
259 unsigned int a[26]={27619871,1391979623,603285685,172424030,219686465,405909358,268293217,2060288893,583249686,1604717347,335537252,1143784024,1278183454,285853801,1363447651,1516758098,2045106584,21459163290,488668660,57259402,1368
260 unsigned int b[11]={425916504,1278353732,1836635897,765213522,1473058898,1864255768,2097193145,2076344587,144101251,169395962,2482253946};
261 lantier aa;
262 aa.p=aa;
263 aa.size=26;
264 lantier bb;
265 bb.p=bb;
266 bb.size=11;
267 lantier res;
268 res=div_euc(aa,bb);
269 Affiche_lantier(res);
270 delete []res.p;
271
272 /* Résultat attendu: {3376975751,2495999155,26374580701,1458211124,4146733654,2911069334,1265901173,1642431167,4278719412,3654881915,2472202807}; */
273
274 Console de débogage Microsoft Visual Studio
275 {3376975751,2495999155,26374580701,1458211124,4146733654,2911069334,1265901173,1642431167,4278719412,3654881915,2472202807}
276 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 55328) avec le code 0.
277 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
278 Appuyez sur une touche pour fermer cette fenêtre.
279
280
281 ueacc1.c:13: C>>p;
282
283
284 // 12 ème test
285
286
287 unsigned int a[14]={910820166,1166539447,417737693,2061804198,297239254,703591460,1277767593,1813997352,601208397,1276253236,147181765,658467799,497386881,1024894579};
288 unsigned int b[6]={115848312,1394191197,689688706,2050701481,1828107761,2168034432};
289 lantier aa;
290 aa.p=aa;
291 aa.size=14;
292 lantier bb;
293 bb.p=bb;
294 bb.size=6;
295 lantier res;
296 res=div_euc(aa,bb);
297 Affiche_lantier(res);
298 delete []res.p;
299
300 /* Résultat attendu: {2360446486,2334079393,421384368,1422921768,3943789293,371577987}; */
301
302 Console de débogage Microsoft Visual Studio
303 {2360446486,2334079392,421384368,1422921769,3943789293,371577987}
304 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 75116) avec le code 0.
305 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
306 Appuyez sur une touche pour fermer cette fenêtre.
307
308
309 ueacc1.c:13: C>>p;
310
311
312 // 13 ème test
313
314
315 unsigned int a[12]={1493609682,1456625770,327547973,1422470612,750154653,496943035,1757240928,311976754,579143443,527777438,1478516201,996881102};
316 unsigned int b[4]={442937880,177575545,1789472563,3867349221};
317 lantier aa;
318 aa.p=aa;
319 aa.size=12;
320 lantier bb;
321 bb.p=b;
322 bb.size=4;
323 lantier res;
324 res=div_euc(aa,bb);
325 Affiche_lantier(res);
326 delete []res.p;
327
328 /* Résultat attendu: {3666275830,3361959922,392444270,729057303}; */
329
330 Console de débogage Microsoft Visual Studio
331 {3666275830,3361959922,392444270,729057303}
332 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 73168) avec le code 0.
333 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
334 Appuyez sur une touche pour fermer cette fenêtre.
335
336
337 ueacc1.c:13: C>>p;
338
339
340 // 14 ème test
341
342
343 unsigned int a[11]={848635161,1589458925,812665111,1345942043,466861856,1971148324,592649592,1356542556,1874366877,265273645,1377053340};
344 unsigned int b[10]={1466736079,643111221,723219375,775878201,970658294,214568997,1526832855,1467681329,1755447269,3955493257};
345 lantier aa;
346 aa.p=aa;
347 aa.size=11;
348 lantier bb;
349 bb.p=b; bb.size=10;
350 lantier res;
351 res=div_euc(aa,bb);
352 Affiche_lantier(res);
353 delete []res.p;
354
355 /* Résultat attendu: {2537412639,2592486335,1541506427,973430149,3440188787,1051501730,3455871045,3542227049,3996796824,964395678}; */
356
357 Console de débogage Microsoft Visual Studio
358 {2537412639,2592486335,1541506427,973430149,3440188787,1051501730,3455871045,3542227049,3996796824,964395678}
359 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 6976) avec le code 0.
360 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
361 Appuyez sur une touche pour fermer cette fenêtre.
```

Citise 1 - GROUPE TD C

compte rendu final

```
// 15 ème test
79
80     unsigned int a[10]={1169042162,89614227,577839039,797313978,449131142,150220964,92899482,683328454,998856126,1681550407};
81     unsigned int b[6]={1415993565,197314521,928615,1239658241,789964113,3504954820};
82     lentier aa;
83     aa.p=a;
84     aa.size=10;
85     lentier bb;
86     bb.p=b;
87     bb.size=6;
88     lentier res;
89     res=div_euci(aa,bb);
90     Affiche_lentier(res);
91     delete []res.p;
92
93
94 /* Résultat attendu: {2512639098,929734849,957992110,121135704,1189708401,2693086096}; */
95
96
97 Console de débogage Microsoft Visual Studio
98 [2512639098,929734849,957992110,1189708401,2693086096]
99
100 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 69484) avec le code 0.
101 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
102 Appuyez sur une touche pour fermer cette fenêtre.
103
104
105 // 16 ème test
106
107     unsigned int a[31]={587080064,285793101,1698348979,1310300239,1061671303,521523625,1308506588,440220510,1989124955,916470209,130746471,1888386079,1052211268,1299788633,637044658,1630050387,2097102603,1066175800,17
108     unsigned int b[16]={1521265848,1382522537,77541826,684882440,21671819,599065451,1992589028,566930702,440706758,761575590,787677173,181609190,1813786858,2087465897,818653848,3443837166};
109     lentier aa;
110     aa.p=a;
111     aa.size=31;
112     lentier bb;
113     bb.p=b;
114     bb.size=16;
115     lentier res;
116     res=div_euci(aa,bb);
117     Affiche_lentier(res);
118     delete []res.p;
119
120
121 /* Résultat attendu: {1910739968,3427623178,3526329092,2035134091,892803386,233991240,464466874,2199863801,3574208397,4010090402,4192226805,3628588589,4266036679,2629578764,3881368249,2933388082}; */
122
123
124 Console de débogage Microsoft Visual Studio
125 [1910739968,3427623178,3526329092,2035134091,892803386,233991240,464466874,2199863801,3574208397,4010090402,4192226805,3628588589,4266036679,2629578764,3881368249,2933388082]
126
127 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 72564) avec le code 0.
128 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
129 Appuyez sur une touche pour fermer cette fenêtre.
130
131
132 // 17 ème test
133
134     unsigned int a[14]={929141142,2078883200,1446850255,1560784892,1654588396,257380779,242259515,1231302208,367569545,18611893,18003544,132429891,240374745,1539269393};
135     unsigned int b[12]={479337878,317916571,75868185,690648063,91698203,2065457213,1352978765,1357688781,682549155,2140655939,1539297971,2496336814};
136     lentier aa;
137     aa.p=a;
138     aa.size=14;
139     lentier bb;
140     bb.p=b;
141     bb.size=12;
142     lentier res;
143     res=div_euci(aa,bb);
144     Affiche_lentier(res);
145     delete []res.p;
146
147
148 /* Résultat attendu: {223336196,542909150,2030224902,2374477927,447131353,1034768665,354656793,3271003521,2711909085,64175495,4036715881,424977781}; */
149
150
151 Console de débogage Microsoft Visual Studio
152 [223336196,542909150,2030224902,2374477927,447131353,1034768665,354656793,3271003521,2711909085,64175495,4036715881,424977781]
153
154 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 72068) avec le code 0.
155 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
156 Appuyez sur une touche pour fermer cette fenêtre.
157
158
159 // 18 ème test
160
161     unsigned int a[17]={1645295884,1970239212,2115297821,426863378,1901558764,1414664428,1987648278,1408663513,1672045288,82424137,492482873,1979614753,1943606036,510485618,1156430886,36497133,2849755011};
162     unsigned int b[4]={1635767956,354413705,2125623196,2331816819};
163     lentier aa;
164     aa.p=a;
165     aa.size=17;
166     lntier bb;
167     bb.p=b;
168     bb.size=4;
169     lntier res;
170     res=div_euci(aa,bb);
171     Affiche_lentier(res);
172     delete []res.p;
173
174
175 /* Résultat attendu: {2949237440,1895619730,1519409480,1298805874}; */
176
177
178 Console de débogage Microsoft Visual Studio
179 [2949237440,1895619730,1519409480,1298805874]
180
181 Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 73964) avec le code 0.
182 Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
183 Appuyez sur une touche pour fermer cette fenêtre.
```

```
// 19 ème test
341
342     unsigned int a[12]=[1937311137,481600861,581662269,1530483428,2028698833,930514635,1463637878,83883357,428236871,1286393442,51697530,855100249];
343     unsigned int b[2]=[1040468559,3613845606];
344
345     lentier aa;
346     aa.p=a;
347     aa.size=12;
348     lentier bb;
349     bb.p=b; bb.size=2;
350     lentier res;
351     res=div_eucl(aa,bb);
352     Affiche_lentier(res);
353     delete []res.p;
354
355 /* Résultat attendu: {1474374818,1268678322}; */
356
357
358 // 20 ème test
359
360     unsigned int a[18]=[990923518,777689008,794130497,823054623,573811396,1304616115,1979484709,610388529,1206887478,1467769018,964722234,1185027026,1652101389,88634314,1084140140,1041928878,570235176,1665802409];
361     unsigned int b[9]=[424928658,443650361,448833396,1888566536,527533718,877870267,1027476331,573231248,3879654164];
362     lentier aa;
363     aa.p=a;
364     aa.size=18;
365     lentier bb;
366     bb.p=b;
367     bb.size=9;
368     lentier res;
369     res=div_eucl(aa,bb);
370     Affiche_lentier(res);
371     delete []res.p;
372
373 /* Résultat attendu: {1040781220,836318477,1191344811,1645576801,669642415,4117350591,1559026176,2785448488,3862855035}; */
374

Console de débogage Microsoft Visual Studio
{1474374818,1268678322}

Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 73324) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.

Console de débogage Microsoft Visual Studio
{1040781220,836318477,1191344811,1645576801,669642415,4117350591,1559026176,2785448488,3862855035}

Sortie de C:\tmp\info1\projet\projet_divison_vf\projet_info\Debug\projet_info.exe (processus 49944) avec le code 0.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.
```

X- Fonction lentier2dec

Rôle : Cette fonction permet de créer une chaîne de caractère contenant l'écriture en base 10 d'un entier à partir de son écriture en base 2^{32} .

Entrée : 1 entier en base 2^{32} .

Sortie : Un pointeur vers une chaîne de caractère contenant l'entier passé en paramètre réécrit en base 10.

Déclaration de fonction : lentier2dec : la fonction (nombre_base_r : 1 lentier) -> 1 pointeur vers une chaîne de caractères

Lexique local :

b10 : 1 pointeur vers caractère
n, length, i : 3 entiers non signés
res_div : 1 quores
l_10n : 1 lentier
j : 1 uint8_t

Algorithme local :

Début

Si ((nombre_base_r.size = 1) ET (nombre_base_r.p[0] = 0))

Alors

b10 <= '0'

Sinon

n <= 9

length <= nombre_base_r.size + 2

res_div.quotient <= nombre_base_r

l_10n <= init_lentier(10^n)

b10 <= Alloue(length * n + 1, caractère)

j <= n

i <= length

Tant que (i > 0)

 res_div <= div_eucl_QR(res_div.quotient, l10n)

 Si ((res_div.quotient.size > 1) OU (res_div.quotient.p[0] ≠ 0) OU
(res_div.reste ≠ 0))

 Alors

 j <= n

 Tant que (j /= 0)

```
b10[(i - 1) * n + j] <= (res_div.reste reste 10) + '0'  
res_div.reste <- res_div.reste div 10  
j <= j - 1  
Fin Tant que  
  
Sinon  
    b10[(i - 1) * n + j] <= '0'  
    i <= 1 -- i valant 1, il est décrémenté à 0 et on sort de la  
boucle (remplace "break;")  
    Fin Si  
    i <= i - 1  
    Fin Tant que  
  
Libère(res_div.quotient.p)  
Libère(l_10n.p)  
b10 <- sAdjust(length * n, b10)  
Fin Si  
  
Fin
```

Rôle : Cette fonction permet de retirer les zéros et les caractères Invalides d'une chaîne de caractères

Entrée : 1 pointeur vers une chaîne de caractères.

Sortie : Un pointeur vers une chaîne de caractère contenant seulement l'entier converti

Déclaration de fonction : sAdjust : la fonction (b : 1 pointeur vers une chaîne de caractères) -> 1 pointeur vers une chaîne de caractères

Lexique local :

```
k      : 1 caractère := b[0]  
i, start : 2 entiers non signés  
d      : 1 pointeur vers caractère
```

Algorithme local :

Début

```
i <= 0  
Tant que ((k < 49) OU (k > 58) ET i <= 1)  
    i <= i + 1  
    k <= b[i]  
Fin Tant que
```

```
d <= Alloue(l - i + 2, caractère)  
d[l - i + 1] <= 'FCH'  
start <= i  
l <= l + i
```

Tant que ($i \leq l$)
 Si ($b[i] > 47$ ET $b[i] < 58$)
 Alors
 $d[i - start] \leq b[i]$
 Sinon
 $i \leq l$ -- Idem, i passe la limite à l'incrément et sort de la boucle
 Fin Si
 $i \leq i + 1$
 Fin Tant que

Libère(b)
 $b \leq d$
Retourner b

Fin

Pour tester la fonction, on utilise un programme principal sur ce modèle :

Lexique principal :

ex_res, n_base_10 : 2 pointeur vers des chaînes de caractères
a : 1 entier
aa : 1 tableau d'entiers

Algorithm principal :

Début

```
aa <- [3197516993, 255446423, 66926]
ex_res <- Alloue(1, caractère)
ex_res[0] <- "1234567891011121314151617"
a.size <- 3
a.p <- aa
n_base_10 <- lentier2dec(a)
Ecrire("Test n0 : ")
Affiche_lentier(a)
Ecrire("Résultat attendu : ", ex_res)
Ecrire("Résultat obtenu : ", n_base_10)
Libère(n_base_10)
```

Fin

Et on l'applique pour chacun des vecteurs de test donné, ce qui donne un programme comme le suivant :

```
int main()
{
    //Lexique {Principal}
    char* n_base_10 = nullptr;
    char* ex_res;
    lentier a;

    /*Test préliminaire*/
    unsigned int aa[3] = { 3197516993, 255446423, 66926 };
    ex_res = (char*)"1234567891011121314151617";
    a.p = aa;
    a.size = 3;
    n_base_10 = lentier2dec(a);
    cout << "Test n0 : ";
    Affiche_lentier(a);
    cout << " Resultat attendu : " << ex_res << endl;
    cout << " Resultat obtenu : " << n_base_10;
    delete[] n_base_10;
    cout << endl << endl << endl;

    /*Test 1*/
    unsigned int aa1[3] = { 1806659176, 1021366019, 527857808 };
    ex_res = (char*)"9737257895872047995341991528";
    a.p = aa1;
    a.size = 3;
    n_base_10 = lentier2dec(a);
    cout << "Test n1 : ";
    Affiche_lentier(a);
    cout << " Resultat attendu : " << ex_res << endl;
    cout << " Resultat obtenu : " << n_base_10;
    delete[] n_base_10;
    cout << endl << endl << endl;
```

• • •

```
/*Test 20*/
unsigned int aa20[4] = { 1201112031, 1461850656, 961260118, 414102093 };
ex_res = (char*)"32808547939433123944033578166106162143";
a.p = aa20;
a.size = 4;
n_base_10 = lentier2dec(a);
cout << "Test n20 : ";
Affiche_lentier(a);
cout << " Resultat attendu : " << ex_res << endl;
cout << " Resultat obtenu : " << n_base_10;
delete[] n_base_10;
cout << endl << endl << endl;

//Fin

system("pause");
return 0;
}
```

Vecteurs de tests:

```
Test n° : (3197516993, 2554646423, 66926)
Resultat attendu : 1234567891011121314151617
Resultat obtenu : 1234567891011121314151617

Test n1 : (1806659176, 1021366819, 5278578808)
Resultat attendu : 9737257895872047995341991528
Resultat obtenu : 9737257895872047995341991528

Test n2 : (1039350890, 1325869069, 1568959855, 1482782840, 345532320, 1176645665, 2195140088, 2013463291, 1010845090, 203523712)
Resultat attendu : 101217071255021195460144859786146819394793035435975051844781714893020196997433772193666754100330
Resultat obtenu : 101217071255021195460144859786146819394793035435975051844781714893020196997433772193666754100330

Test n3 : (1628937191, 72205753, 1078503725)
Resultat attendu : 1989488210792754735282766679
Resultat obtenu : 1989488210792754735282766679

Test n4 : (1706894180, 2081063073, 908661504, 676840878, 1146065784, 832542319, 1426475006, 1055216528, 368077969, 855338803, 876362068)
Resultat attendu : 187189816446218547954596411380882293657868920283883649688702577890113289261518671821514171269546763108
Resultat obtenu : 187189816446218547954596411380882293657868920283883649688702577890113289261518671821514171269546763108

Test n5 : (514513261, 1897728888, 1509287715, 1233910532)
Resultat attendu : 97760464185199810690926976331837659851
Resultat obtenu : 97760464185199810690926976331837659851

Test n6 : (687673136, 655386739, 124894522)
Resultat attendu : 2303897286357151826839408432
Resultat obtenu : 2303897286357151826839408432

Test n7 : (1832032404)
Resultat attendu : 1832032404
Resultat obtenu : 1832032404

Test n8 : (899185100, 695393047, 5470922243, 2046932510)
Resultat attendu : 1621747015681185772131816038356667908660
Resultat obtenu : 1621747015681185772131816038356667908660

Test n9 : (620137996, 9779952587, 564316072, 179548528, 911532912, 1472977576, 856389398, 2057598697, 158937248, 135380757, 527015216, 999718849, 594209998, 1508445123)
Resultat attendu : 1096396328971808177281980469528945377857495130075406881213590927988348893906461986024610394460762129200818887469547411360746160083406942713056788
Resultat obtenu : 109639632897180817728198046952894537785749513007540688121359092798834889390646198603469547411360746160083406942713056788

Test n10 : (344454438, 870249190, 591659335, 1134049768)
Resultat attendu : 89848679329281957073936071279380002086
Resultat obtenu : 89848679329281957073936071279380002086

Test n11 : (1247046074, 1258944290, 443644135, 931594831, 1680352821, 1342829235, 1626988678)
Resultat attendu : 1021277345609082868891511743274797695934386014161539815361745283514
Resultat obtenu : 1021277345609082868891511743274797695934386014161539815361745283514

Test n12 : (1242278897, 180833668, 623939413, 72747037, 220668492, 803487941, 984279949, 1693646069, 1659877340)
Resultat attendu : 1922006512193964289804493779362209122267518082489179696668763686113824814830288707281
Resultat obtenu : 1922006512193964289804493779362209122267518082489179696668763686113824814830288707281

Test n13 : (1851683317, 1795258097, 612242928, 231214885, 638493289, 1296452926, 1739660009)
Resultat attendu : 1092002863239941603023622966050701529717128806771467738769847644661
Resultat obtenu : 1092002863239941603023622966050701529717128806771467738769847644661

Test n14 : (1550907364, 462445551, 587901427, 8372473484, 2028347878, 1824047501, 1798417774, 218508265, 610868684, 1252286048, 1659327600, 98563714, 1256088364)
Resultat attendu : 4949240159245122786251127345585267349147381281734483122787289562258449957577412461885985655668570770451694676373154027900884
Resultat obtenu : 4949240159245122786251127345585267349147381281734483122787289562258449957577412461885985655668570770451694676373154027900884

Test n15 : (1902399782, 1880627777)
Resultat attendu : 8874657819688980874
Resultat obtenu : 8874657819688980874

Test n16 : (2123068275, 53602071, 1811159093, 1669230969, 48425763, 5588070386, 1373430365, 1843683860, 117031314, 1604645250, 334693501, 229282592, 1196821611, 330935593, 1780189956)
Resultat attendu : 129301996678896908412908358276575033465873385359275492492262899480771964093236771493169064581932576654186867589864826767688822901766661459
Resultat obtenu : 129301996678896908412908358276575033465873385359275492492262899480771964032367714931690645819325766541868675898640267676888222901766661459

Test n17 : (918837028, 170179799, 1532111393, 006308873, 1966597567, 184861784, 1223539558, 1071400867, 1368473718, 1323923272, 180005583, 2114605759)
Resultat attendu : 19399381269090284134611209551221663479161276273941846342254956887934532074183554437961217653744557966920503285532
Resultat obtenu : 19399381269090284134611209551221663479161276273941846342254956887934532074183554437961217653744557966920503285532

Test n18 : ([2060033861, 794001198, 1054424034, 48576586, 57617082, 497007547, 184617082, 1940961447, 193207759, 1368473718, 1323923272, 180005583, 2114605759]
Resultat attendu : 2681072781142219882956402726721925568351614578864534851593377239337359884223974293692317537267247672616778617298407571161688915628830545
Resultat obtenu : 2681072781142219882956402726721925568351614578864534851593377239337359884223974293692317537267247672616778617298407571161688915628830545

Test n19 : (2115348176, 1777673873, 70745332, 1499975921, 236491098)
Resultat attendu : 80473750702011952452662690157637500818194540240
Resultat obtenu : 80473750702011952452662690157637500818194540240

Test n20 : ([2101112031, 1461858656, 961260118, 414182993])
Resultat attendu : 3288054793943123944033578166106162143
Resultat obtenu : 3288054793943123944033578166106162143
```

Pour écrire cette fonction, l'idée de base était de récupérer chaque chiffre du nombre final, en commençant par le moins significatif. En prenant le reste par 10 du nombre donné, on obtient ce chiffre que l'on place dans la chaîne de caractère, puis il faut calculer le quotient par 10 du nombre pour pouvoir accéder au second chiffre de la même façon, et on répète ces étapes jusqu'à la fin du nombre donné.

La première idée pour améliorer cet algorithme primitif vient de la manière dont sont calculés le quotient et le reste par la fonction en question. En effet, dans cette fonction, le quotient et le reste sont calculés "simultanément", et une fois le calcul terminé, le quotient est ignoré et le reste est retourné à la fonction appelante. Or, si on pouvait récupérer le quotient et le reste en même temps, il n'y aurait plus que deux étapes au lieu de 3 : on calcule le

reste/quotient, on copie la valeur du reste dans la chaîne, puis recalcule le quotient/reste, etc...

Nous avons donc dû créer un type “quores” composé d'un entier base r pour le quotient et d'un entier non signé pour le reste (le reste est plus petit que le diviseur, donc <10) et également une copie de la fonction de division euclidienne (div_eucl_QR : la fonction(a : 1 entier, b : 1 entier) -> 1 quores) qui renvoie le quotient et le reste d'un coup.

Inspirés par les TP, le fait que les opérations sur un entier base r seront bien plus longues que sur des entiers non signés nous a donné l'idée de ne pas diviser par 10, mais par 1000000000 (10^9 est la plus grande puissance de 10 de taille 1 en base r, la division étant bien plus optimisée pour des opérations avec un diviseur de taille 1) mais le reste étant non plus le chiffre voulu mais les 9 chiffres suivants, nous avons dû rajouter une boucle intérieur qui agit comme expliqué premièrement pour récupérer chaque chiffre, mais en effectuant, pour chaque boucle principale 1 opération en base r, et 9 opérations en base 2.

Il nous fallait maintenant déterminer les paramètres des boucles. Pour la boucle secondaire, on est sûrs de faire 9 itérations à chaque boucle principale, sauf pour la dernière, mais finir ces 9 opérations à la dernière boucle ne risque que de rajouter des 0 dans la chaîne, donc on gardera cette valeur de 9 constamment. Pour la boucle principale, il faut jusqu'à 1 itération de plus que le nombre de cases dans le nombre base r (on divise par 10^9 , mais la valeur max dans une case est 4.2×10^9 , donc si on divise jusqu'à la dernière case, il se peut que la valeur de cette case aie encore jusqu'à 4 comme valeur, il faut donc la re-diviser. Comme l'indice dans cette boucle est non signé, il pourra varier de nombre_base_r.size + 2 à 1 (car lorsqu'il vaut 0, on sort de la boucle).

Un dernier problème se posait alors : la chaîne de caractère était pleine de 0, de valeurs n'étant même pas des nombres mais pas de caractère de fin de chaîne (CFC, ou '\0'). Pour éviter d'encombrer plus la fonction, nous en avons créé une seconde, sAdjust (inspirée de lAdjust, mais pour une “string”). Cette fonction prends comme paramètres le nombre de caractères supposés être dans la chaîne, ladite chaîne (un pointeur vers celle-ci) et parcours dans un premier temps la chaîne de caractère de la fin au début (des nombres les plus significatifs vers les moins significatifs) jusqu'à trouver une valeur étant un nombre non nul, puis copie le reste des valeurs et place un caractère de fin de chaîne au bout.

Enfin, cette fonction cherchant des valeurs non nulles dans la chaîne, une chaîne supposée ne contenir que 0 ne peut pas être vérifiée par cette fonction. Pour éviter ce cas unique, on vérifie au début de la première fonction si, pour le nombre en base r passé en paramètre, la longueur est de 1, et la première case contient 0. Si c'est le cas, on crée une chaîne ne contenant que 0 et le caractère de fin de chaîne et on renvoie le résultat.

Ainsi, la fonction testée retourne bien des chaînes de caractères correctes.

XI- Organisation et conclusion

Pour conclure, la majorité du projet nous semble réussie. À l'exception de la division euclidienne, toutes les fonctions fonctionnent parfaitement dans la mesure de nos tests. Pour ce qui est de la division, son comportement est étrange et imprévisible : tous les vecteurs de tests sont bien fonctionnels, ils n'engendrent pas d'erreurs et renvoient bien les bonnes valeurs. Cependant, malgré les efforts de l'équipe responsable, dès que l'on essaie de calculer des restes prévisibles même sur des grands nombres (à partir de nombres répétés ou de grande puissances de 10) les résultats sont erronés...

La division étant une pierre angulaire des fonctions de calculs modulaires, ces autres fonctions sont par défaut incapable de produire des résultat adéquats pour des grands diviseurs, rendant la fonction `exp_mod` inutilisable pour des grands exposants. Cependant les résultats intermédiaires, hors calculs de restes, sont fonctionnels.

Pour ce qui est de l'organisation de groupe, celle-ci ne s'est vraiment faite qu'à partir de la première date de rendu des premières fonctions. Quelques jours après cette date, le besoin d'une mise en commun s'est fait sentir pour permettre aux équipes de niveaux avancés d'avoir accès aux fonctions de base dont ils avaient besoin. Pour éviter que chaque équipe n'ai sa propre version des fonctions, agrémentée à leur besoin et que le rendu final ne soit trop désorganisé, il a fallu trouver une solution simple d'accès, autant pour récupérer les fichiers nécessaires que pour appliquer ou proposer des changements.

Ainsi fut créée la zone de dépôse GitHub. Accessible aussi bien en lecture qu'en écriture par tous ceux qui en demandaient l'accès. L'utilisation de GitHub nous a aussi permis plus de flexibilité dans nos apports personnels avec un historique complet des versions disponible à tout moment, faisant qu'aucun changement n'était vraiment permanent ou irréparable. Très rapidement les premières fonctions (`Affiche_lentier`, `IAdjust`, `IAdjust_realloc`, `mult_classique`, `Add_lentier`, `cmp_lentier` et `sub_lentier`) ont été regroupées, et testées pour vérifier si une fois mises en commun ces fonctions étaient toujours totalement fonctionnelles.

Rapidement, les fonctions qui étaient les plus avancées ou rapides à écrire une fois le premier rendu effectué ont aussi rejoint le dépôt, `mul_mod`, `dec2lentier` et `lentier2dec` notamment, car elles étaient déjà avancées mais en pause en attendant la fonction de division euclidienne, qui, étant la plus longue et plus complexe à traiter fut l'une des dernières à être "fonctionnelle" au point de pouvoir avancer et tester les fonctions de calcul modulaire et `lentier2dec`.

Une fois ce squelette réuni, s'en est suivi environ une semaine d'aller-retours et de débogage, aussi bien en équipe, qu'entre les équipes, qui ont permis de repérer et de corriger de nombreuses petites erreurs, principalement dans la gestion des tailles qui devenaient parfois nulles, et des l'accès aux tableaux en dehors de leurs bornes qui causaient des erreurs lors de la suppression des pointeurs. Désormais, tous nos tests sur la version actuelle n'ont pas retourné d'erreurs de compilation, de génération, ou d'exécution, en dehors des résultats de la division qui ne sont pas corrects.

Projet réalisé par le groupe de TD C de Télécom Saint-Etienne:

Affiche_lentier, IAdjust, IAdjust_realloc : - LAVEDRINE Louis
- PETIT Johan
- BONNET Loïs

Mult_classique, Mul_mod : - CHAPTAL Lilian
- GREFFE Anaïs.

Add_lentier : - MARCANDELLA Axel
- BOURGEY Pierre

cmp_lentier, sub_lentier : - CIZERON Oscar
- TROUILLOT Sophie
- MARTIN Jean-Baptiste.

exp_mod : - SINNO Emilie
- Mechkour Ghali
- MOUTTE Paul.

dec2lentier : - LE BARS Mattéo
- MONDON Emile
- EZZAR Ahmed.

div_eucl : - AUTECHAUD Simon
- BERNARD Romaric
- LKHAILIDI Othmane.

lentier2dec : - SAUZE Ewan
- ADAM Maxime
- PINTO Loris.

Chef de projet : M. Florent BERNARD

FIN DE PROJET
